COGTEACHBENCH: A STANDARDIZED LEARNER-STATE BENCHMARK FOR EVALUATING LLM TUTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating an LLM tutor requires a learner whose misconception the tutor must repair, yet current tutoring benchmarks simulate learners with role-prompted LLMs that accept corrections readily, retain every teacher message, and can copy answers from context. We introduce **CogTeachBench**, a benchmark that evaluates LLM tutors by whether they repair a learner's misconception, using learner proxies designed to counter these artifacts. An acceptance control (α) counters post-training compliance, and a cognitive-load control (β), calibrated on students' load ratings of 557 teaching segments, limits what the learner retains from overloaded teacher turns. The benchmark contains 316 oracle-verified misconception-repair scenarios built from LiveCodeBench problems released after the learner models' pretraining cutoff. It scores tutors with four metrics, namely Cognitive Bottleneck Clearance Score (CBCS) for post-teaching repair, Adaptive ZPD Span (AZS) for ability sensitivity, Process-level Scaffolding Quality (PSQ), and Pedagogical Cognitive Efficiency (PCE) for load-normalized gain. Across five tutors, the CogTeachBench ranking matches the ordering of human transfer success in a dialogue-replay study with 36 undergraduates (Spearman $\rho = 1.00$) and is largely preserved when the learner model family changes ($\rho = 0.90$). The four metrics identify strengths that a single score would conflate. Gemini 3.1 Pro achieves the strongest post-teaching clearance, whereas GPT-5.4 achieves the highest load-normalized efficiency with comparable process quality. The learner profiles further reveal tutors that end the lesson when a high-acceptance learner signals agreement before the repair is executable.

1 INTRODUCTION

Large language models (LLMs) are increasingly used as tutors, and a tutor succeeds when its instruction lets a learner remove a pre-existing misconception. This requires diagnosis, guided revision, and support for independent recovery in addition to knowledge of the solution, so measuring it requires a learner whose state the tutor must change. Coding benchmarks evaluate problem solving, and education-oriented benchmarks often score isolated responses or rubric judgments. Recent tutoring benchmarks add multi-turn interaction with LLM-simulated learners but specify the learner mainly through role prompts, coarse ability labels, or static error descriptions. The LLM's own behavior then fills in whatever the prompt leaves unspecified. It accepts plausible corrections readily, retains every teacher message in full, may solve the task from memory, and can copy code from its context, so measured gains partly reflect these artifacts.

We introduce **CogTeachBench**, a benchmark for evaluating LLM tutors through controlled misconception repair. CogTeachBench instantiates this protocol in algorithmic programming because the domain provides executable validation, concrete learner artifacts, and structural checks for answer leakage. Starting from LiveCodeBench problems released after the learner-proxy models' pretraining cutoff (Jain et al., 2025), we construct misconception-bearing scenarios, remove cases that learners can repair without instruction, and retain only cases that become repairable under structured oracle guidance. The final benchmark contains 316 oracle-verified scenarios, each centered on a learner misconception, a buggy artifact or reasoning trace, and a diagnostic key for the target bottleneck.

The core of CogTeachBench is its learner proxy. We identify the artifacts that an LLM brings to learner role-play and counter each one by construction or control (Table 1). Post-cutoff problems

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and a zero-intervention filter rule out repair from memory or unaided competence. Post-training makes an LLM proxy readily accept plausible corrections, so the α mechanism controls whether the learner accepts, resists, or requests stronger evidence from the teacher. In-context access lets a proxy retain an entire teacher message, far beyond what a learner can hold, so the β mechanism limits the information retained from cognitively overloaded teacher turns. Overload is decided by a load model fitted to students' ratings, with thresholds calibrated separately on a weaker and a stronger student group for the 3B and 7B proxies. Anti-spooning checks and hidden executable post-tests keep copied answers and fluent agreement from counting as repair. Without these controls, the same LLM expresses confusion in 0.65% of turns and outputs code immediately in 30.09%. With them, it shows the confusion, slowdown requests, and resistance that cognitive-load theory predicts under overload. The main evaluation therefore uses the four profiles in which both controls are active.

We evaluate tutoring with both gain- and process-oriented metrics. **CBCS** measures whether instruction clears the learner's misconception-centered bottleneck relative to the pre-teaching state; **AZS** measures whether this gain depends on learner ability; **PSQ** evaluates process-level scaffolding quality; and **PCE** measures teaching gain per unit of teacher-imposed cognitive load. On five tutors, the CogTeachBench ranking matches the ordering of human transfer success (Spearman $\rho = 1.00$), and the four metrics select different best tutors. Overall, CogTeachBench contributes (i) standardized learner proxies that counter documented artifacts of LLM role-play, with overload triggers calibrated to student load ratings; (ii) a controlled misconception-repair formulation with 316 oracle-verified scenarios and integrity checks; and (iii) an empirical analysis showing that teaching gain, process quality, ability sensitivity, and cognitive efficiency expose different strengths of LLM tutors.

2 Related Work

2.1 Educational and Tutoring Benchmarks for LLMs

Educational evaluation of LLMs has increasingly moved from general task completion toward education-specific settings. Early benchmarks in this direction focus on static educational tasks, including grade-level adaptation, curriculum-oriented question answering, hierarchical cognitive assessment, pedagogical knowledge, and teacher-role safety (Naeem et al., 2025; Ma et al., 2025; Zhang et al., 2025; Lelièvre et al., 2025; Jiang et al., 2026). A second line of work evaluates tutoring behavior more directly by assessing teacher responses in explanation, feedback, hint generation, mistake correction, and open-ended pedagogical dialogue (Macina et al., 2023; Maurya et al., 2025; Srinivasa et al., 2025; Macina et al., 2025; Xu et al., 2025). More recent work further introduces multi-turn or outcome-oriented protocols, where teacher models interact with simulated learners or are evaluated by student performance changes after instruction (Li et al., 2026; Mandyam, 2025; Ma et al., 2024). Despite these advances, existing educational and tutoring benchmarks still often rely on isolated responses, rubric-based judgments, or task-level performance signals, leaving limited evidence about how learner improvement, teaching process quality, and student-side simulation jointly shape the validity of tutoring evaluation (Li et al., 2024; Hardy & Kim, 2026).

2.2 Pedagogical Quality Criteria for Learning-Oriented Evaluation

Teaching quality is typically defined through process-level behaviors that support learner understanding alongside answer correctness. Classic work on tutoring emphasizes scaffolding, contingent support, and the gradual transfer of responsibility during problem solving (Wood et al., 1976), while studies of human and intelligent tutoring systems highlight explanation, feedback, questioning, misconception repair, and adaptive response to learner behavior as central components of effective instruction (Chi et al., 2001; Vanlehn, 2006). Conceptual change theory further stresses that learners often need to confront inconsistencies between prior beliefs and task evidence before revising misconceptions (Posner et al., 1982). Cognitive load theory and working-memory research provide a complementary perspective, suggesting that instructional support should be calibrated to the learner's processing capacity (Sweller, 1988; Cowan, 2001). Recent LLM tutor evaluations have begun to translate these ideas into operational criteria, including adaptive explanation, feedback quality, hint generation, pedagogical dimension taxonomies, open-ended tutoring quality, and Bloom-based explanation assessment (Srinivasa et al., 2025; Maurya et al., 2025; Macina et al., 2025; Zoumpoulidi et al., 2025). These lines of work indicate that learning-oriented evaluation should account for diagnosis, scaffolding, cognitive conflict, and cognitive appropriateness.

2.3 Learner Proxies and Student Simulation in Tutoring Benchmarks

Student-side modeling has become increasingly important in tutoring benchmarks, since the evaluation of a tutor depends on the learner state being simulated. Existing work commonly represents learners through written mistakes, static problem contexts, or LLMs prompted to play students with specified errors or cognitive levels (Macina et al., 2023; Li et al., 2026; Mandyam, 2025; Yue et al., 2025). These designs make tutoring evaluation more interactive, but the learner is usually specified only through a role prompt or a coarse label, and the LLM's own behavior fills in the rest. Studies of such prompt-specified learners document recurring artifacts. They generate overly advanced responses when asked to simulate imperfect learners (Wu et al., 2025), broadly capable models face a competence paradox when emulating partially knowledgeable students (Yuan et al., 2026), and strong problem-solving ability does not ensure reliable estimation of student struggles (Li et al., 2025). Wu et al. (2025) and Yuan et al. (2026) respond by constraining the simulation rather than abandoning LLM learners, and related work calls for standardized learners with structured error states, ability variation, and cognitively grounded constraints (Jiang & Zhang, 2025; Salhan et al., 2026). CogTeachBench builds its learner proxies around these requirements.

3 The Construction of CogTeachBench

3.1 Misconception-Repair Scenario Design

The core component of CogTeachBench is a set of misconception-repair scenarios. Each scenario is built around an algorithmic programming problem and stores: (i) an initial learner query that presents the learner's symptoms and locally confident self-report without directly naming the root cause, and that may tactically misattribute the failure to a superficially plausible but incorrect suspect when supported by the buggy artifact, (ii) a buggy program consistent with that misunderstanding, (iii) a misconception rationale that initializes the learner-side cognitive state, (iv) a diagnostic key used for curation, oracle verification, and evaluation metadata, and (v) a verified reference solution and executable tests. During formal teacher–learner interaction, the teacher receives the problem, buggy artifact, learner self-report, and verified reference solution, but not the misconception rationale or diagnostic key. The teacher is therefore assessed on whether it can diagnose and remediate the learner's misconception during interaction.

To cover different levels of instructional difficulty, scenarios are organized into three misconception-weight buckets. Lower-weight cases typically involve local repair targets, while higher-weight cases involve broader reasoning mechanisms or problem-solving paradigms. The weight label also constrains construction. Each candidate scenario must preserve a clear relation between the injected misconception, the buggy artifact, and the diagnostic key, which keeps unrelated implementation errors out of the intended bottleneck.

3.2 Programming Context and Teacher-Facing Inputs

To exclude memorized problem solving by the learner proxies, CogTeachBench sources its base programming problems from LiveCodeBench snapshots released after the pretraining cutoff of the learner-proxy models (Jain et al., 2025). The teacher-facing context includes the original problem statement, the learner's erroneous attempt, the learner's self-report, and a verified reference solution. The reference solution aligns teachers on task-solving ability, so the benchmark measures diagnostic and pedagogical behavior. Under this solution-aware setting, which mirrors tutoring systems supplied with trusted answer keys, how a teacher uses the known solution without collapsing learner agency is itself part of teaching competence. At the same time, the teacher is not given the diagnostic key; it must infer the learner's bottleneck from the flawed artifact and self-report, and direct or paraphrased answer transfer is constrained by the anti-spooning protocol and downstream executable post-test.

3.3 Data Generation and Curation

The construction pipeline follows a multi-stage expert-seeded, LLM-expanded, and expert-curated process. We begin with 239 cleaned programming problems and obtain 198 verified tasks whose reference solutions pass the corresponding tests. Human experts then write 36 seed misconception schemes, with 12 seeds for each of the three misconception-weight buckets. Each seed specifies

the intended cognitive bottleneck, plausible learner belief, expected failure pattern, and naturalness boundary.

We use a multi-model LLM expansion procedure to generate task-conditioned misconception candidates from the expert seeds. The expansion pool uses Claude Opus 4.5, Gemini 3 Pro Preview, and GPT-5.2 with the same seed-conditioned generation prompt. For each seed, the expansion procedure produces 30 candidate misconception plans, yielding 1080 candidate plans in total. Human experts review all generated candidates according to a fixed guideline, checking weight alignment, learner-side plausibility, causal consistency with the buggy artifact, diagnostic value, implementability, and non-redundancy. Accepted candidates are realized as misconception-bearing scenarios, and 546 realized scenario candidates enter zero-intervention screening. The seed-expansion prompt and expert review guideline are reported in Appendix C.1 and Appendix C.2.

We then conduct zero-intervention screening, where a learner model attempts to repair the misconception without teacher assistance. Scenarios that can be self-repaired are removed, since they do not represent stable instructional bottlenecks. This screening retains 365 scenario candidates and covers 166 unique tasks. Finally, we conduct oracle teachability verification. Each retained scenario candidate is paired with structured oracle guidance that provides fault localization and corrective information. A lower-ability learner model then attempts the post-test under this oracle condition. Scenarios that remain unsolved even under oracle guidance are discarded. This stage yields 316 oracle-verified teaching scenarios, which form the final benchmark asset.

4 Evaluation Methodology

To evaluate LLM tutors in educational settings, CogTeachBench uses a component-aligned protocol that measures instruction-induced teaching gain while controlling original task solving. Following Hardy & Kim (2026), surface process ratings alone are insufficient because fluent language can be misaligned with the intended learning outcome, while outcome-only scoring can hide direct answer transfer or poor scaffolding. The protocol therefore separates learner improvement, ability-tier generalization, scaffolding quality, and teacher-imposed cognitive load.

4.1 Evaluation Components

Misconception-repair outcome. The outcome component measures whether teaching changes the learner's executable behavior from its pre-teaching failure state to post-teaching success (Hardy & Kim, 2026; Li et al., 2026; Mandyam, 2025). Direct answer transfer counts as an integrity violation, and the anti-spooning check guards CBCS without serving as a separate teaching-quality metric. It targets answer-level code transfer, so natural-language explanations, formulas, and conceptual hints are flagged only when they become executable or structurally solution-equivalent code. Its semantic rules are expert-specified, while its structural weights and thresholds are grid-searched to maximize F_1 on 150 expert-labeled turns; details are in Appendix H.

Standardized learner-profile conditions. The learner component tests whether a teacher remains effective under controlled learner variation, using proxies that counter the artifacts in Table 1. We instantiate twelve learner profiles by crossing two ability tiers with six α/β conditions: baseline, α-high, α-low, β-only, α-high+β, and α-low+β (Table 18). The two ability tiers are standardized checkpoints whose ordering is verified at the benchmark boundary, where the 7B proxy repairs 10 of 49 scenarios that the 3B proxy cannot under identical oracle guidance (Appendix B.2), and whose overload thresholds are calibrated on the weaker and stronger student groups, respectively (Appendix G.2). This follows calls for diverse, cognitively inspired student proxies (Wu et al., 2025; Li et al., 2025; Salhan et al., 2026; Yuan et al., 2026). The four profiles with both controls active form the main evaluation setting; the remaining eight serve as ablations isolating each control. Load computation and subjective-load calibration, selection of the F_{abstract} classifier by agreement with three-expert annotations, and parameter-effect checks are reported in Appendices G, I.1, and F.

Process-quality component. Because post-test gain alone does not distinguish guided repair from lucky recovery, CogTeachBench also scores whole-dialogue tutoring quality. The process component checks whether the teacher identifies the root concept, keeps the learner cognitively active, uses evidence or traces, and slices repair into manageable steps, following cognitive diagnosis, conceptual change, contingent tutoring, and cognitive-load theory (Leighton & Gierl, 2007; Posner et al., 1982;

Table 1: Artifacts of LLM learner role-play and the control CogTeachBench applies to each. Evidence is detailed in Section 3 and Appendices F, G.2, and H.

LLM role-play artifact	Control	Evidence
Solving the task from memory	Post-cutoff problems	Sec. 3
Repair without instruction	Zero-intervention filter	181 of 546 candidates removed
Post-training compliance	α acceptance control	Defensive dialogues 50.3% (α-high) vs. 58.3% (α-low), $p = 0.002$
Full-context retention	β overload control, student-calibrated	Confused turns 0.65% $\rightarrow$ 9.5–29.1%; trigger AUC 0.85/0.89
Copying answers from context	Anti-spooning check	$F_1 = 0.837$ against expert labels
Agreement without repair	Hidden executable post-test	CBCS decided by tests, not by an LLM judge

Chi et al., 2001; Wood et al., 1976; Vanlehn, 2006; Sweller, 1988). The compact rubric is in Table 31. PSQ is scored with DeepSeek-V4-Pro (DeepSeek-AI, 2026) under a structured dialogue-judge prompt, selected by agreement with three-expert annotations as reported in Appendix I.3.

4.2 METRICS

Cognitive Bottleneck Clearance Score (CBCS). The primary outcome is teaching gain. It records whether a learner who failed before instruction can pass an independent post-teaching test after interacting with the teacher. For case i, formal evaluation uses a fixed `fix@3` post-test protocol. Let $R_i = 1$ if at least one of the three post-teaching repair attempts passes the executable tests and no answer-transfer flag is raised; otherwise $R_i = 0$. CBCS is the misconception-bucket-normalized gain rate:

$$\text{CBCS} = \frac{\sum_{\omega \in \Omega} \lambda_\omega \left(\frac{1}{|B_\omega|} \sum_{i \in B_\omega} R_i \right)}{\sum_{\omega \in \Omega} \lambda_\omega}, \tag{1}$$

where Ω is the set of misconception-weight buckets, B_ω is the evaluated case set in bucket ω, and λ_ω is the calibrated bucket weight; calibration details are reported in Appendix E. CBCS follows the teaching-gain framing of Hardy & Kim (2026). The benchmark measures learner improvement after instruction while preserving cognitive-diagnostic structure through misconception buckets (Leighton & Gierl, 2007; Al-Hossami & Bunescu, 2026).

Adaptive ZPD Span (AZS). AZS measures ability-tier sensitivity by comparing a teacher's clearance under matched α/β profile conditions:

$$\text{AZS} = \text{CBCS}_{\text{7B}} - \text{CBCS}_{\text{3B}} \tag{2}$$

where the two learners share the same α/β condition. AZS is signed. Values near zero indicate low ability-tier dependence, positive values mean higher CBCS with the 7B learner, and negative values mean higher CBCS with the 3B learner. AZS is a scaffolding-adaptation measure, consistent with ZPD accounts of support that must remain within the learner's reachable range (Vygotsky, 1978; Wood et al., 1976).

Process-level Scaffolding Quality (PSQ). PSQ measures whole-dialogue scaffolding quality on four normalized dimensions: root-cause concept attribution (RCA), Socratic guidance ratio (SGR), execution trace and cognitive conflict (ETC), and micro-scaffolding slice (MSS). These dimensions are grounded in cognitive diagnosis and conceptual change (Leighton & Gierl, 2007; Posner et al., 1982), tutoring and guided-discovery research (Chi et al., 2001; Vanlehn, 2006; Wood et al., 1976), and cognitive-load-aware scaffolding (Vygotsky, 1978; Sweller, 1988). For record i,

$$\mathcal{D} = \{\text{RCA}, \text{SGR}, \text{ETC}, \text{MSS}\}, \quad \text{PSQ}_i = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{s_{i,d}}{S_d^{\max}}, \quad \text{PSQ} = \frac{1}{|\mathcal{J}|} \sum_{i \in \mathcal{J}} \text{PSQ}_i \,. \tag{3}$$

where $s_{i,d}$ is the dialogue-judge score for dimension d, $S_d^{\max}$ is the maximum score for that dimension, and $\mathcal{J}$ is the set of records with valid judge outputs. Normalizing each dimension before averaging prevents any single rubric scale from dominating PSQ. The PSQ judge uses structured, low-temperature scoring with dimension-wise consistency checks, following evidence and cautions from LLM-judge evaluation work (Zheng et al., 2023; Li et al., 2024). DeepSeek-V4-Pro is selected from

candidate judge models by agreement with three-expert annotations on sampled dialogue records; details are in Appendix I.3.

Pedagogical Cognitive Efficiency (PCE). PCE evaluates whether outcome gains are achieved with manageable teacher-imposed load. Let L_i be the summed load of teacher turns in record i, and $\bar{L}$ its λ_ω-weighted mean over records with load annotations:

$$\text{PCE} = \frac{\text{CBCS}}{\bar{L}}. \tag{4}$$

Higher PCE indicates more teaching gain per unit of teacher-imposed cognitive load. The underlying load model combines instruction demand, formal/code complexity, and abstraction level. Educational experts specify the feature-level scoring rules; the three combination weights are fitted by cross-validated simplex search on 557 student-rated segments, which also calibrate the overload thresholds. The abstraction-level component is implemented with Qwen3-8B. Student-rating calibration and judge-selection details are reported in Appendix G and Appendix I.1, and sensitivity to the scaffolding credits in Appendix G.3.

4.3 Evaluated Models and Setup

We evaluate six teacher conditions. GPT-5.4 (OpenAI, 2026), Qwen3.5 Plus, and Qwen3.5 Flash (Qwen Team, 2026) are run on all twelve profiles in Table 18. Gemini 3.1 Pro (Google DeepMind, 2026) and Claude Opus 4.6 (Anthropic, 2026) are run on the four full-control profiles, which combine high or low acceptance with overload for both the 3B and 7B learner proxies. Qwen2.5-7B Instruct (Qwen Team, 2024), the 7B learner model itself, serves as a self-teaching reference on the two 7B full-control profiles. Full per-condition run metadata and results are reported in Appendix K.

5 Results

We report the four full-control profiles as the main comparison and use the remaining profiles of the 12-profile runs as ablations. Full per-profile results and trajectories are in Appendix K and Appendix K.1. The four metrics select different top tutors, and the resulting ranking carries over to another learner family ($\rho = 0.90$) and to human learners ($\rho = 1.00$). The learner-side controls further show how tutors differ in scaffolding, robustness to cognitive load, and adaptation to learner ability.

Table 2: Main results on the full-control profiles, with stratified bootstrap 95% confidence intervals. CBCS, PSQ, and PCE are averaged over the available full-control profiles; AZS is the signed mean over matched 3B/7B pairs. Qwen2.5-7B Instruct is the self-teaching reference.

Teacher	Profiles	CBCS↑	AZS→ 0	PSQ↑	PCE↑
Gemini 3.1 Pro	4	0.581 [0.513, 0.650]	0.047 [-0.027, 0.123]	0.779 [0.751, 0.804]	0.190 [0.158, 0.230]
GPT-5.4	4	0.509 [0.444, 0.573]	0.109 [0.020, 0.200]	0.793 [0.775, 0.812]	0.253 [0.212, 0.299]
Claude Opus 4.6	4	0.486 [0.427, 0.544]	0.032 [-0.059, 0.122]	0.666 [0.642, 0.690]	0.112 [0.093, 0.133]
Qwen3.5 Plus	4	0.466 [0.405, 0.529]	0.035 [-0.060, 0.129]	0.695 [0.675, 0.715]	0.084 [0.068, 0.102]
Qwen3.5 Flash	4	0.313 [0.257, 0.370]	0.073 [-0.004, 0.149]	0.616 [0.590, 0.641]	0.057 [0.044, 0.072]
Qwen2.5-7B Instruct	2	0.062 [0.033, 0.093]	–	0.355	0.014

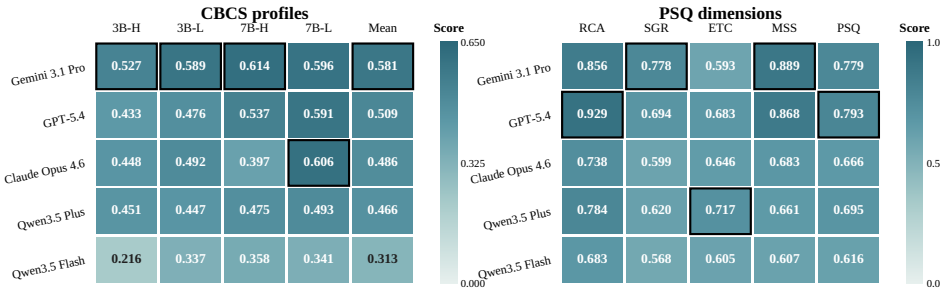


Figure 1: CBCS and PSQ dimensions on the full-control profiles. Left: CBCS by profile, where H/L denotes α-high/low and the mean averages the four profiles. Right: normalized PSQ dimension means on the same profiles; black-bordered cells mark column maxima.

5.1 Outcome, Process Quality, and Efficiency Select Different Tutors

Teaching gain, judged scaffolding, and load efficiency produce different leaderboards. On the full-control profiles, Gemini 3.1 Pro achieves the highest CBCS (0.581), followed by GPT-5.4 (0.509), Claude Opus 4.6 (0.486), Qwen3.5 Plus (0.466), and Qwen3.5 Flash (0.313). A paired case-level bootstrap separates Gemini 3.1 Pro from GPT-5.4 (difference 0.072; 95% CI [0.014, 0.131]) and Qwen3.5 Plus from Qwen3.5 Flash (difference 0.153; 95% CI [0.094, 0.214]); details are in Appendices E.2 and E.3.

Process and efficiency measures change the ordering. GPT-5.4 achieves the highest PCE (0.253) and a PSQ statistically tied with Gemini 3.1 Pro (0.793 vs. 0.779). The self-teaching reference shows that the scenarios require teaching beyond the learner's own competence, since Qwen2.5-7B Instruct, the 7B learner model itself, reaches a CBCS of 0.062.

5.2 Teacher Rankings Transfer Beyond the Learner Proxy

The teacher ranking is not specific to one learner family. Replacing the Qwen learners with Llama-3.2-3B and Llama-3.1-8B under the α-low+β profile largely preserves it (Spearman $\rho = 0.90$); only GPT-5.4 and Claude Opus 4.6 swap, and their CBCS intervals overlap. It also carries over to human learners. In a dialogue-replay study, 36 undergraduates who had not taken an algorithms course read the recorded dialogues and then solved a structurally different transfer task. Their success rates order the five teachers exactly as CBCS does ($\rho = 1.00$, from 84.7% for Gemini 3.1 Pro to 54.2% for Qwen3.5 Flash), and the ordering remains nearly unchanged on each participant's first task alone ($\rho = 0.975$, with one tie). Details are in Appendix B.

5.3 Process Quality Is Multi-Dimensional

The right panel of Figure 1 decomposes process quality by dimension. GPT-5.4 obtains the highest aggregate PSQ (0.793) and the strongest RCA score (0.929), indicating strong root-cause concept attribution. Gemini 3.1 Pro is close in aggregate PSQ (0.779) and leads on SGR (0.778) and MSS (0.889), suggesting stronger guided participation and smaller repair slices. Qwen3.5 Plus obtains the highest ETC score (0.717). Because these behaviors are not interchangeable, the PSQ dimensions give each tutor a process profile in addition to an aggregate score.

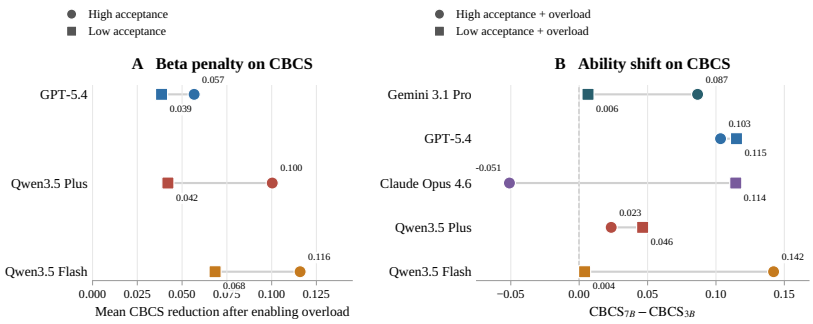


Figure 2: Learner-profile effects on teaching gain. Left: beta penalty, measured as the average CBCS reduction after enabling overload relative to the matched acceptance-only condition; lower values indicate stronger robustness to cognitive-load constraints. Right: ability shift on the full-control profiles, measured as $\mathrm{CBCS}_{7B} - \mathrm{CBCS}_{3B}$; values closer to zero indicate lower ability-tier dependence.

5.4 Cognitive-Load Constraints Expose Robustness Differences

Learner-profile controls materially change the measured teaching effect. In the full 12-profile runs, enabling β lowers CBCS relative to matched β-off profiles for all three teachers with complete coverage. Under α-high, the drops are 0.057 for GPT-5.4, 0.100 for Qwen3.5 Plus, and 0.116 for Qwen3.5 Flash; under α-low, the drops are 0.039, 0.042, and 0.068. The effect is largest for Qwen3.5 Flash and smallest for GPT-5.4.

Figure 2 quantifies these profile-control effects directly. The left panel shows that the Qwen3.5 Flash penalty is largest under high acceptance. The PCE column in Table 2 shows the corresponding efficiency pattern, in which GPT-5.4 preserves the strongest load-normalized gain under overload and Qwen3.5 Flash the weakest among the external tutors. The β control thus tests whether a teacher can pace, localize, and sequence repair without overloading the learner. Full trajectories are in Appendix K.1.

5.5 Learner Ability and Acceptance Change the Teaching Signal

AZS identifies which tutors depend on learner ability (Figure 2, right). GPT-5.4 is the only tutor whose AZS interval excludes zero (0.109; 95% CI [0.020, 0.200]), so its clearance rises with the stronger learner. Gemini 3.1 Pro is nearly ability-stable under α-low (0.006), and Claude Opus 4.6 teaches the 3B learner more effectively under α-high (-0.051). Ability sensitivity is therefore a property of the tutor, which an aggregate CBCS would hide.

This profile dependence changes how tutor behavior should be interpreted. Higher-ability or high-acceptance learners may provide fluent agreement signals before executable repair has been verified, while lower-acceptance or lower-ability learners expose remaining gaps more directly. Appendix J provides paired logs supporting this mechanism.

6 Discussion

Tutor evaluation requires plural metrics. CBCS, PSQ, and PCE select different best tutors. Outcome-only evaluation can reward a teacher that produces post-test repair without revealing whether the interaction was well scaffolded, and rubric-only evaluation can reward a polished dialogue whose effect on the learner remains uncertain. Efficiency adds a further constraint, because a tutor that compresses too much information into each turn may be less usable for learners under load. Tutor benchmarks should therefore report gain, process, and efficiency jointly.

Learner simulation is part of measurement. Learner proxies directly shape the tutoring signal being measured. They determine what evidence the teacher receives and how much resistance the teacher must handle. Aggregate CBCS can therefore hide whether a tutor is robust across learner

ability tiers and acceptance profiles; profile-conditioned scores, including AZS, keep this dependence visible. We therefore validate the proxy at three levels. Its overload trigger is calibrated to students' load ratings, its behavior moves away from LLM artifacts toward the confusion and resistance that cognitive-load theory predicts, and its teacher ranking carries over to another learner family and to human transfer success. This mirrors how LLM judges were established (Zheng et al., 2023), through agreement with humans at the level the benchmark reports rather than identity with humans at every turn. Standardized learner proxies thus play a role similar to standardized patients in medical education (Barrows, 1993), exposing every tutor to the same controlled misconception, acceptance, and load conditions. Unlike a human learner, whose state changes irreversibly once taught, a proxy can be reset with a fresh context, so all tutors are compared on an identical learner state, a within-learner design that human studies cannot provide (Appendix B.1).

Apparent agreement can mislead. The agreement-without-repair cases explain one mechanism behind the profile effects. Under α-high or 7B profiles, learners are more likely to produce fluent agreement signals, which can make a teacher stop before the misconception has been repaired. The broader lesson is that an effective tutor should maintain an evolving model of the learner's cognitive state. This model should track the active misconception, the evidence that has actually changed the learner's reasoning, and whether apparent agreement reflects conceptual repair or conversational compliance. For LLM-tutor research, this shifts attention from generating helpful explanations to estimating and updating learner state throughout the interaction.

Load-aware tutoring. The load model and both overload thresholds that trigger β are fitted to students' ratings, and the retained content follows a fixed head–tail rule specified with educational experts, which applies the same retention rule to every teacher. The role of β is to expose whether a tutor depends on dense single-turn explanations that become brittle when the learner cannot retain the whole message. Tutoring systems should not maximize explanation coverage alone. They should pace disclosure, localize repair steps, and verify retention before adding more material.

7 Conclusion

We present CogTeachBench, a benchmark that evaluates whether LLM tutors guide learner proxies from pre-existing misconceptions to executable correction. It combines oracle-verified misconception-repair scenarios, learner proxies that counter documented artifacts of LLM role-play, executable post-teaching tests, process-quality scoring, cognitive-load calibration, and anti-spooning checks. Its teacher ranking matches human transfer success in a dialogue-replay study, and its four metrics show that tutoring quality is multi-dimensional. Gemini 3.1 Pro achieves the strongest post-teaching clearance, and GPT-5.4 achieves the highest load-normalized efficiency with process quality on par with Gemini 3.1 Pro. LLM tutor evaluation can therefore rest on learner-conditioned evidence of misconception repair, scaffolding quality, and cognitive efficiency. Limitations are discussed in Appendix A.

AI use statement

We used generative AI tools to assist in drafting and polishing the manuscript, to check grammar, to assist in coding the project, and to assist in generating the benchmark assets.

Ethics statement

This work adheres to the ICLR Code of Ethics. It includes two low-risk studies with human participants: a questionnaire in which 13 student volunteers rated the cognitive load of 557 teaching segments (Appendix G.1), and a dialogue-replay transfer study with 36 CS undergraduates (Appendix B.1). In the transfer study, participants read prerecorded teaching dialogues and completed programming transfer tests; they did not interact with the evaluated models, and teacher identities were hidden. Participation in both studies was voluntary and uncompensated. Participants gave informed consent and could withdraw at any time. No sensitive personal information was collected, and responses were anonymized before analysis. Formal IRB approval or equivalent institutional review was not obtained. The benchmark is derived from public LiveCodeBench problems, and released load-rating records contain only de-identified scores.

Reproducibility statement

We release the CogTeachBench assets and pipeline code in an anonymous repository: https://anonymous.4open.science/r/ICLR-2027-CogTeachBench-72EE. The assets include the 316 executable misconception-repair scenarios with hidden tests, diagnostic metadata, and learner-profile configurations. The code covers teacher–learner interaction, anti-spooning checks, load scoring, and metric computation, and provides a single benchmark entry point for evaluating a new teacher model. We also release audit records, including oracle-screen records for all 316 retained and 49 rejected variants, de-identified load ratings for the 557 calibration segments, and dialogue trajectories with anti-spooning evidence and load traces. Because the full trajectory files exceed the file-size limit of the hosting platform, we provide compressed and condensed versions for inspection. Model identifiers, API routes, and decoding parameters are listed in Appendix D, and all runtime prompts are given in Appendix L.

References

Erfan Al-Hossami and Razvan Bunescu. McMining: Automated Discovery of Misconceptions in Student Code. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 160–178. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-short.10. URL https://aclanthology.org/2026.eacl-short.10/.

Anthropic. Introducing Claude Opus 4.6, February 2026. URL https://www.anthropic.com/news/claude-opus-4-6. Accessed 2026-05-06.

Howard S. Barrows. An overview of the uses of standardized patients for teaching and evaluating clinical skills. *Academic Medicine*, 68(6):443–451, 1993.

Michelene T.H. Chi, Stephanie A. Siler, Heisawn Jeong, Takashi Yamauchi, and Robert G. Hausmann. Learning from human tutoring. *Cognitive Science*, 25(4):471–533, 2001. doi: 10.1207/s15516709cog2504_1. URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog2504_1.

Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, 2001. doi: 10.1017/S0140525X01003922. URL https://www.cambridge.org/core/product/identifier/S0140525X01003922/type/journal_article.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Google DeepMind. Gemini 3, 2026. URL https://deepmind.google/models/gemini/. Accessed 2026-05-06.

Michael Hardy and Yunsung Kim. Knowledge without Wisdom: Measuring Misalignment between LLMs and Intended Impact, 2026. URL http://arxiv.org/abs/2603.00883.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Jianxiao Jiang and Yu Zhang. Student Development Agent: Risk-free Simulation for Evaluating AIED Innovations, 2025. URL http://arxiv.org/abs/2510.09183.

Yilin Jiang, Mingzi Zhang, Xuanyu Yin, Sheng Jin, Suyu Lu, Zuocan Ying, Zengyi Yu, and Xiangjie Kong. EduGuardBench: A holistic benchmark for evaluating the pedagogical fidelity and adversarial safety of llms as simulated teachers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(37):31356–31364, 2026. doi: 10.1609/aaai.v40i37.40399. URL https://ojs.aaai.org/index.php/AAAI/article/view/40399.

Jacqueline Leighton and Mark Gierl. *Cognitive diagnostic assessment for education: Theory and applications*. Cambridge University Press, 2007.

Maxime Lelièvre, Amy Waldock, Meng Liu, Natalia Valdés Aspillaga, Alasdair Mackintosh, María José Ogando Portela, Jared Lee, Paul Atherton, Robin A. A. Ince, and Oliver G. B. Garrod. Benchmarking the pedagogical knowledge of large language models, 2025. URL https://arxiv.org/abs/2506.18710.

Ming Li, Han Chen, Yunze Xiao, Jian Chen, Hong Jiao, and Tianyi Zhou. Can LLMs Estimate Student Struggles? Human-AI Difficulty Alignment with Proficiency Simulation for Item Difficulty Prediction, 2025. URL http://arxiv.org/abs/2512.18880.

Zhen Li, Xiaohan Xu, Tao Shen, Can Xu, Jia-Chen Gu, Yuxuan Lai, Chongyang Tao, and Shuai Ma. Leveraging Large Language Models for NLG Evaluation: Advances and Challenges. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16028–16045. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.896. URL https://aclanthology.org/2024.emnlp-main.896.

Zheng Li, Siyao Song, Jingyuan Ma, Rui Li, Ying Zeng, Minghao Li, and Zhifang Sui. TeachBench: A Syllabus-Grounded Framework for Evaluating Teaching Ability in Large Language Models, 2026. URL http://arxiv.org/abs/2601.21375.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *Advances in Neural Information Processing Systems*, 37:74325–74362, 2024. doi: 10.52202/079017-2365. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html.

Guoqing Ma, Jia Zhu, Hanghui Guo, Weijie Shi, Yue Cui, Jiawei Shen, Zilong Li, and Yidan Liang. EduEval: A hierarchical cognitive benchmark for evaluating large language models in chinese education, 2025. URL https://arxiv.org/abs/2512.00290.

Jakub Macina, Nico Daheim, Sankalan Chowdhury, Tanmay Sinha, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. MathDial: A Dialogue Tutoring Dataset with Rich Pedagogical Properties Grounded in Math Reasoning Problems. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5602–5621. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.372. URL https://aclanthology.org/2023.findings-emnlp.372.

Jakub Macina, Nico Daheim, Ido Hakimi, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. MathTutorBench: A Benchmark for Measuring Open-ended Pedagogical Capabilities of LLM Tutors. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 204–221. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.11. URL https://aclanthology.org/2025.emnlp-main.11/.

Aishwarya Mandyam. TutorTest: Evaluating Language Model-based Tutoring Policies Using Surrogate Tasks. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=UoJZv2YfDX.

Kaushal Kumar Maurya, Kv Aditya Srivatsa, Kseniia Petukhova, and Ekaterina Kochmar. Unifying AI Tutor Evaluation: An Evaluation Taxonomy for Pedagogical Ability Assessment of LLM-Powered AI Tutors. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1234–1251. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.57. URL https://aclanthology.org/2025.naacl-long.57/.

Numaan Naeem, Abdellah El Mekki, and Muhammad Abdul-Mageed. EduAdapt: A Question Answer Benchmark Dataset for Evaluating Grade-Level Adaptability in LLMs, 2025. URL http://arxiv.org/abs/2510.17389.

OpenAI. GPT-5.4 thinking system card, March 2026. URL https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf. Accessed 2026-05-06.

George J. Posner, Kenneth A. Strike, Peter W. Hewson, and William A. Gertzog. Accommodation of a scientific conception: Toward a theory of conceptual change. *Science Education*, 66(2):211–227, 1982. doi: 10.1002/sce.3730660207. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sce.3730660207.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Suchir Salhan, Andrew Caines, and Paula Buttery. Pedagogical alignment of LLMs requires diverse cognitively-inspired student proxies. In *First Workshop on CogInterp: Interpreting Cognition in Deep Learning Models*, 2026. URL https://openreview.net/forum?id=1XL8DtZiFe.

Rakshith S. Srinivasa, Zora Che, Chen Bo Calvin Zhang, Diego Mares, Ernesto Hernandez, Jayeon Park, Dean Lee, Guillermo Mangialardi, Charmaine Ng, Ed-Yeremai Hernandez Cardona, Anisha Gunjal, Yunzhong He, Bing Liu, and Chen Xing. TutorBench: A Benchmark To Assess Tutoring Capabilities Of Large Language Models, 2025. URL http://arxiv.org/abs/2510.02663.

John Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12 (2):257–285, 1988. doi: 10.1207/s15516709cog1202_4. URL https://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1202_4.

Kurt Vanlehn. The Behavior of Tutoring Systems. *Int. J. Artif. Intell. Ed.*, 16(3):227–265, 2006.

Lev S Vygotsky. *Mind in society: The development of higher psychological processes*, volume 86. Harvard university press, 1978.

D. Wood, J. S. Bruner, and G. Ross. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, and Allied Disciplines*, 17(2):89–100, 1976. doi: 10.1111/j.1469-7610.1976.tb00381.x.

Tao Wu, Jingyuan Chen, Wang Lin, Mengze Li, Yumeng Zhu, Ang Li, Kun Kuang, and Fei Wu. Embracing Imperfection: Simulating Students with Diverse Cognitive Levels Using LLM-based Agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9887–9908. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.488. URL https://aclanthology.org/2025.acl-long.488/.

LiQing Xu, Qiwei Li, Tianshuo Peng, Zuchao Li, Hai Zhao, and Ping Wang. Can large language models be good language teachers? In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 23957–23971, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1222. URL https://aclanthology.org/2025.emnlp-main.1222/.

Zhihao Yuan, Yunze Xiao, Ming Li, Weihao Xuan, Richard Tong, Mona Diab, and Tom Mitchell. Towards Valid Student Simulation with Large Language Models, 2026. URL http://arxiv.org/abs/2601.05473.

Liang Yue, Yihong Tang, Kehai Chen, Jie Liu, and Min Zhang. MASTER: Enhancing large language model via multi-agent simulated teaching. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=5GaDcRVgBw.

Min Zhang, Hao Chen, Wenqi Zhang, Didi Zhu, Xin Lin, Bo Jiang, Aimin Zhou, Fei Wu, and Kun Kuang. OmniEduBench: A comprehensive chinese benchmark for evaluating large language models in education, 2025. URL https://arxiv.org/abs/2510.26422.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Maria-Eleni Zoumpoulidi, Eleni Batsi, Georgios Paraskevopoulos, Vassilis Katsouros, and Alexandros Potamianos. Bloomxplain: A framework and benchmark dataset for pedagogically sound LLM-generated explanations based on bloom's taxonomy. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025. URL https://openreview.net/forum?id=EMMPng2LLl.

A Limitations

This work has several limitations. First, CogTeachBench is currently instantiated in algorithmic programming, where executable post-tests and structural leakage checks make learner repair highly verifiable. Extending the framework to other domains will require domain-specific post-tests, misconception generators, teachability filters, and leakage validators. Second, the present protocol is text-only and does not evaluate diagrams, visual simulations, interactive widgets, or executable demonstrations that multimodal tutoring systems may use. Third, the learner proxies are validated at the level of overload triggers, behavioral direction, and teacher ranking, but not at the level of individual learning trajectories. The human study replays recorded dialogues rather than letting students interact with the tutors, and the retention rule applied after overload is expert-specified. CogTeachBench should therefore be viewed as a controlled testbed for comparing teaching policies under standardized learner conditions. Claims about how much a particular student would learn from a particular tutor still require interactive human studies. If these limitations are ignored, benchmark scores may be over-interpreted as evidence of real-world tutoring readiness, which could mislead educational deployment or obscure the needs of diverse human learners.

B Human Transfer and Learner-Family Checks

B.1 Human Dialogue-Replay Transfer Study

This study tests whether the teaching gains that CBCS measures on learner proxies transfer to human learners at small scale. It recruited 36 CS undergraduates who had completed an introductory programming course but had not taken an algorithms course. We used 20 w_2 scenarios, fixing the misconception bucket to reduce between-bucket heterogeneity, and prepared prerecorded teaching dialogues from all five benchmark teachers. For each task, participants read the buggy code and misconception rationale, reviewed the teaching dialogue, and then completed a structurally different transfer post-test within 15 minutes without revisiting the dialogue. Hidden executable tests determined success. Participants did not interact with the teacher models directly, and teacher identities were hidden throughout the study.

The misconception rationale is the learner-side field that initializes the learner proxy's cognitive state; it is distinct from the initial learner query, which is the only learner-authored text the teacher sees. Because the study replays dialogues recorded with learner proxies, participants must enter the same initial state as the proxy that the dialogue was addressed to. Giving them the rationale instils the same preconceived incorrect belief directly. Without it, participants would spend part of the session reading the problem and forming their own interpretations, which would differ across participants and from the proxy's state. The replay therefore tests whether a teacher's dialogue repairs a fixed, shared misconception rather than whatever understanding each participant happened to construct.

The study used a within-participant design. Each participant completed ten distinct tasks and saw each teacher twice; each selected task was paired with one teacher for that participant, and no participant completed the same transfer test twice. Teacher order was counterbalanced and task assignment was balanced. Each task was completed by 18 participants and evaluated with every teacher: for each task, three teachers received four observations and two received three, with the allocation rotated across tasks. This yields 360 task-level observations, 72 per teacher.

Table 3: Benchmark CBCS and human transfer success in the dialogue-replay study.

Teacher	Benchmark CBCS (rank)	Human transfer success (rank)
Gemini 3.1 Pro	0.581 (1)	61/72 (84.7%, 1)
GPT-5.4	0.509 (2)	57/72 (79.2%, 2)
Claude Opus 4.6	0.486 (3)	55/72 (76.4%, 3)
Qwen3.5 Plus	0.466 (4)	51/72 (70.8%, 4)
Qwen3.5 Flash	0.313 (5)	39/72 (54.2%, 5)

The human ordering matches the CBCS ordering exactly, with Spearman $\rho = 1.000$ and Pearson $r = 0.991$. Success ranges from 54.2% to 84.7%, a 30.6 percentage-point difference between the first and fifth teachers.

Performance on later tasks may benefit from programming concepts or problem-solving strategies learned from earlier, different tasks. We therefore repeat the analysis using only each participant's first observation, before any concept could carry over from an earlier task. Counterbalancing assigned the first position to seven or eight participants per teacher.

Table 4: First-observation transfer success, using only each participant's first task.

Teacher	Benchmark CBCS (rank)	First-observation success (rank)
Gemini 3.1 Pro	0.581 (1)	6/7 (85.7%, 1)
GPT-5.4	0.509 (2)	6/8 (75.0%, 2)
Claude Opus 4.6	0.486 (3)	5/7 (71.4%, 3 tied)
Qwen3.5 Plus	0.466 (4)	5/7 (71.4%, 3 tied)
Qwen3.5 Flash	0.313 (5)	3/7 (42.9%, 5)

The first-observation results retain the benchmark pattern. Gemini 3.1 Pro and GPT-5.4 remain first and second, Claude Opus 4.6 and Qwen3.5 Plus tie in the middle, and Qwen3.5 Flash remains last. Using midranks for the tie gives Spearman $\rho = 0.975$, and the Pearson correlation between CBCS and success rate is $r = 0.994$. The first-to-fifth gap is 42.9 percentage points.

Design constraints of human tutoring studies. Several features of this design are compromises imposed by human learners. A participant's state changes irreversibly once they have been taught: after one dialogue has repaired a misconception, the same participant cannot be returned to the pre-teaching state and taught the same misconception by a different teacher, which is why each participant completes each task only once, with a single teacher. Different tasks also share programming concepts, so a participant's later tasks are not independent of earlier ones, which is why we report the first-observation analysis separately. As a result, a human study cannot compare several teaching strategies on the same learner state; it can only compare them across different participants and tasks, at a cost of recruitment time and limited sample size. Learner proxies remove this constraint. A proxy can be reset to an identical initial state with a fresh context, so every teacher is evaluated on exactly the same learner state for every scenario and profile, a within-learner comparison that is infeasible with humans. We regard this as a central reason why standardized learner proxies are needed for AI-for-education evaluation, with human studies such as this one serving as external validation rather than as the primary comparison instrument.

After the study, participants described some dialogues as difficult to follow because of uneven pacing, weak decomposition, limited guidance, excessive information, and long responses that obscured the main solution line. They also mentioned reasoning aloud and revisions that reversed an earlier direction. These comments agree with the weaknesses in localized repair and step sequencing observed for Qwen3.5 Flash in the benchmark.

B.2 Oracle-Boundary Rerun with the 7B Learner

The oracle-teachability screen removed 49 scenarios that the 3B learner could not repair under structured oracle guidance (9, 14, and 26 in w_1, w_2, and w_3; Table 14). We reran both learners

on all 49 scenarios, holding the oracle guidance, prompts, hidden tests, and decoding fixed. The rerun produced 0/49 repairs for 3B and 10/49 for 7B (20.4%; 95% CI [11.5%, 33.6%]; exact paired $p = 0.001953$). The 7B learner therefore repairs cases beyond the 3B boundary under identical guidance, so the two learners are not interchangeable at the benchmark boundary. In the teaching benchmark, the effect of this difference is teacher- and profile-dependent, with full-control AZS ranging from -0.051 to $+0.142$.

B.3 Learner-Family Replacement

We replaced the Qwen2.5-3B/7B-Instruct learners with the architecturally distinct Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct checkpoints. The replacement uses the α-low+β-on condition, which exercises both evidence demand and overload-sensitive retention. All five teachers used the same formal-benchmark task set, prompts, anti-spooning checks, and post-test pipeline; only the learner checkpoint changed, giving ten runs in total.

Table 5: Teacher CBCS under Qwen and Llama learner families in the α-low+β-on condition, averaged over the two checkpoints of each family. Ranks are within each learner family.

Teacher	Qwen CBCS (rank)	Llama CBCS (rank)
Gemini 3.1 Pro	0.593 (1)	0.548 (1)
GPT-5.4	0.534 (3)	0.496 (2)
Claude Opus 4.6	0.549 (2)	0.478 (3)
Qwen3.5 Plus	0.470 (4)	0.427 (4)
Qwen3.5 Flash	0.339 (5)	0.293 (5)

The teacher-rank correlation is Spearman $\rho = 0.90$. Gemini 3.1 Pro remains first, and Qwen3.5 Plus and Qwen3.5 Flash remain fourth and fifth; only GPT-5.4 and Claude Opus 4.6 exchange positions, and their full-control CBCS intervals overlap substantially (Table 15). Llama-based CBCS is uniformly lower by 0.038–0.071, a family-level shift in absolute difficulty that leaves the rest of the teacher ranking unchanged.

C Benchmark Construction Details

CogTeachBench starts from 239 cleaned LiveCodeBench tasks and obtains 198 verified answer-generation successes before misconception injection. Misconception candidates are constructed through an expert-seeded, LLM-expanded, and expert-curated process. Human experts first write 36 seed misconception schemes, including 12 seeds for each of the three misconception-weight buckets. A multi-model LLM expansion pool then expands the expert seeds into task-conditioned candidate plans using the same seed-conditioned prompt. The pool consists of Claude Opus 4.5, Gemini 3 Pro Preview, and GPT-5.2. For each seed, the expansion procedure produces 30 candidate plans, yielding 1080 candidate plans in total. All candidate plans are reviewed by human experts before scenario realization. After expert review and realization, 546 scenario candidates enter zero-intervention screening. Zero-intervention screening retains 365 candidates across 166 unique tasks, removing cases that the learner can repair without instruction. White-box oracle verification then confirms 316 candidates as teachable under structured expert guidance; these form the final full benchmark asset with stable identifiers `ctb_1`–`ctb_316`. The zero-intervention stage allows up to five independent attempts, each with up to six repair rounds; the first two attempts begin with a reflection-only turn before code generation. Oracle teachability is checked with the 3B learner at temperature 0.1 under fix@1 sampling.

Table 6: Misconception-candidate construction and filtering pipeline.

Stage	Count	Description
Cleaned tasks	239	LiveCodeBench tasks after preprocessing.
Verified tasks	198	Tasks whose reference solutions pass the tests.
Expert seed schemes	36	12 seeds for each misconception-weight bucket.
Multi-model LLM-expanded plans	1080	30 task-conditioned plans per seed from a three-model expansion pool.
Expert-reviewed scenarios	546	Accepted candidates realized as learner queries, buggy artifacts, rationales, and diagnostic keys.
After zero-intervention screening	365	Candidates not repaired by the learner without teacher assistance.
Final oracle-verified scenarios	316	Candidates repairable under structured oracle guidance.

C.1 Expert-Seeded Misconception Expansion

Each expert seed specifies a target misconception weight, a seed title, a seed description, the intended cognitive bottleneck, naturalness boundaries, and rejection conditions. The multi-model LLM expansion step is used only to increase candidate diversity under these expert-defined constraints. It does not determine whether a candidate enters the benchmark. All three expansion models use the same seed-conditioned prompt, and generated candidates are pooled before human expert review. Human experts make the acceptance decision after reviewing the expanded candidates.

Seed-Based Misconception Candidate Expansion Prompt

```
System:
You are a professional Python algorithms teacher who has worked with many students
 carrying natural misconceptions.

You design misconception-injection candidates for Python programming benchmark tasks.

The input includes an expert-written seed misconception scheme. The seed defines the
 intended cognitive bottleneck, misconception weight, and naturalness boundary. Your
 job is to expand the seed into task-conditioned misconception candidates. You should
  preserve the pedagogical intent of the seed, but you should not merely paraphrase
 or duplicate it. Each candidate must be adapted to the given programming task and
 reference solution.

Work in strict JSON only.

For the requested seed:
- propose up to the requested number of distinct, realistic misconception-injection
 candidates
- if fewer realistic candidates exist, return only the realistic ones
- if none are realistic for this task, return status NO with a concise rejection
 reason
- naturalness is mandatory: every candidate must read like a misconception a real
 student could plausibly hold after sincerely attempting the task
- do not force a candidate by inventing a bug that only exists to satisfy the seed or
  weight label
- if the misconception would require unnatural edits, implausible student beliefs, or
  a contrived implementation path, return status NO

Hard constraints:
1. Preserve the task interface and problem semantics.
2. The future buggy code must be syntactically valid Python 3.
3. The bug must be a teachable logic bug, not a syntax error, formatting issue,
 import issue, or random implementation slip.
4. Each candidate must explicitly model the student's mistaken belief, local
 reasoning chain, and why the misconception feels natural for this task.
5. Each implementation plan must be concrete enough for a later model call to realize
  directly.
6. The plan should stay close to the reference solution structure unless the seed
 explicitly targets a deeper paradigm mistake.
7. The candidate must remain aligned with the seed's misconception weight and
 cognitive-bottleneck type.
8. Prefer rejecting the seed-task combination over proposing an artificial or
 shoehorned misconception.
```

```
9. Do not include code snippets, quoted examples, backticks, regex literals, markdown
 , bullets, or prose outside the JSON object.
10. Use double quotes for all JSON keys and string values.
11. Return exactly one JSON object.

User:
Misconception weight:
{seed.weight}

Seed title:
{seed.title}

Seed description:
{seed.description}

Seed cognitive bottleneck:
{seed.cognitive_bottleneck}

Seed naturalness boundary:
{seed.naturalness_boundary}

Seed rejection conditions:
{seed.rejection_conditions}

Task question:
{task["question"]}

Excellent reference solution that passes all test cases and is provided for you to
 study before designing the misconception:
{reference_solution}

Candidate generation task:
- Generate up to {candidate_quota} distinct misconception-injection candidates for
 this task based on the expert seed.
- Each candidate must instantiate the seed in a task-specific way.
- Candidates should differ in the learner belief, failure chain, or implementation
 manifestation.
- If only a few realistic candidates exist, return only those candidates.
- If no realistic candidate exists, return status NO.

Output discipline:
- Return exactly one JSON object.
- Use double quotes everywhere.
- Use multi-sentence strings when needed; do not compress complex reasoning into one
 short line.
- Do not include code snippets, quoted examples, backticks, regex literals, markdown,
  comments, bullets, or any prose outside the JSON object.

Return exactly this JSON schema:
{
  "status": "implementable or NO",
  "seed_alignment_summary": "...",
  "candidates": [
    {
      "misconception_summary": "...",
      "student_model": {
        "mistaken_belief": "...",
        "why_it_feels_right": "...",
        "hidden_blind_spot": "..."
      },
      "failure_chain": {
        "triggering_context": "...",
        "wrong_inference": "...",
        "resulting_behavior": "..."
      },
      "implementation_plan": {
        "target_code_region": "...",
        "edit_strategy": "...",
        "must_preserve": "...",
        "expected_failure_mode": "..."
      },
      "weight_alignment_rationale": "...",
      "naturalness_rationale": "...",
      "difference_from_seed": "..."
    }
  ],
  "rejection_reason": null
}
```

C.2 HUMAN EXPERT REVIEW MANUAL

The following manual was provided to human experts when reviewing LLM-expanded misconception candidates. The manual was designed to support consistent review decisions while allowing experts to revise plausible candidates when minor corrections were sufficient.

Human Expert Review Manual for Misconception Candidates

Reviewer role. Thank you for reviewing candidate misconception plans for an algorithmic-programming tutoring benchmark. Each candidate was generated from an expert-written seed and a specific programming task. The purpose of the review is to determine whether the candidate can be realized as a natural, teachable, and diagnostically useful benchmark scenario.

A valid scenario should represent a misconception that a real learner could plausibly hold, that can be expressed through a buggy but syntactically valid Python solution, and that can be diagnosed and repaired through tutoring without directly revealing the final answer.

Input provided for each review item. For each candidate, please review the following materials: 1. the task statement; 2. the reference solution; 3. the expert seed, including its intended weight, title, cognitive bottleneck, naturalness boundary, and rejection conditions; 4. one LLM-expanded misconception candidate, including the proposed learner belief, failure chain, implementation plan, and naturalness rationale.

The task and reference solution may be used to judge whether the proposed misconception is plausible and implementable. A candidate should not be accepted only because it can produce a failing program. The failure should be meaningfully caused by the stated misconception.

Review procedure. Please review each candidate according to the following steps.

Step 1: Identify the root misconception. Please summarize, in one sentence, what the learner incorrectly believes. If the candidate does not provide a clear learner belief, or if it only describes a code edit without a learner-side belief, it should be marked as REJECT.

Step 2: Check alignment with the seed. Please determine whether the candidate preserves the cognitive intent of the expert seed. The candidate does not need to copy the seed literally, but it should instantiate the same kind of misconception in the current task. A candidate should be marked as REJECT if it changes the seed into a different error type, matches the seed only superficially, or forces the requested weight onto an unrelated bug.

Step 3: Check the weight bucket. Please use the following operational definitions.

Low-weight misconception. The error concerns a local repair target, such as a boundary condition, comparison direction, update timing, indexing convention, or local condition. A learner can usually repair it after noticing one concrete counterexample or one local trace.

Medium-weight misconception. The error concerns a nontrivial reasoning step, invariant, state update rule, control-flow assumption, or interaction between multiple local components. A learner usually needs to revise a piece of reasoning rather than only changing one local expression.

High-weight misconception. The error concerns the broader algorithmic strategy, state representation, recurrence formulation, decomposition of the problem, proof idea, or global interpretation of the task. A learner usually needs to reorganize the solution logic rather than patching a small local line.

If the candidate is plausible but appears to belong to a different weight bucket, please mark it as REVISE and record the corrected weight. If the candidate's difficulty mainly comes from code length, accidental implementation complexity, obscure syntax, or arbitrary edits rather than a cognitive misconception, it should be marked as REJECT.

Step 4: Check learner-side naturalness. Please judge whether a sincere learner could plausibly arrive at this belief after reading the task, examples, or common algorithmic patterns. Plausible sources of naturalness include overgeneralizing from examples, applying a familiar template in the wrong context, trusting a locally valid invariant that fails globally, confusing similar state definitions, or misreading an implicit constraint.

A candidate should be marked as REJECT if the learner belief is incoherent, adversarial, intentionally careless, based on ignoring the task entirely, or only exists because a generator was asked to create a bug. A candidate should also be rejected if its naturalness rationale merely states that the bug is common without explaining why a learner would reasonably believe it.

Step 5: Check causal consistency. Please verify that the candidate forms a coherent chain: learner belief $\rightarrow$ wrong inference $\rightarrow$ implementation choice $\rightarrow$ expected failure behavior.

A candidate should be marked as REJECT if the implementation plan does not follow from the learner belief, if the same code edit could occur as a random slip without the stated misconception, or if the expected failure mode is unrelated to the proposed belief.

Step 6: Check implementation feasibility. Please determine whether the candidate can be realized as a syntactically valid Python 3 program while preserving the original task interface. The realized bug should fail meaningful tests, but the program should remain close enough to the intended solution structure to support targeted tutoring.

A candidate should be marked as REJECT if it requires changing the input-output format, changing the problem semantics, deleting essential parts of the solution for no pedagogical reason, introducing syntax errors, relying on undefined behavior, or creating a program that fails for reasons unrelated to the misconception.
Step 7: Check tutoring suitability. Please judge whether a tutor could diagnose and repair the misconception from the learner query and buggy artifact. The scenario should support a natural learner query that reports symptoms, local confidence, and possibly a plausible but incorrect self-diagnosis, without directly naming the hidden root cause.
A candidate should be marked as REJECT if the only reasonable way to help the learner is to reveal the full reference solution, if the root cause cannot be inferred from the artifact, or if the interaction would become ordinary debugging without a stable misconception.
Step 8: Check non-redundancy. Please compare the candidate with already accepted candidates for the same task and weight. Near-duplicate candidates should be merged or rejected. A candidate should be kept only if it represents a materially different learner belief, failure chain, or implementation manifestation.
Decision labels. Please assign exactly one final decision.
ACCEPT: The candidate satisfies all mandatory checks. Minor wording edits are allowed, but the learner belief, failure chain, weight, and implementation plan are already usable.
REVISE: The candidate is potentially valid, but one or more fixable issues should be corrected before realization. This label is appropriate for wrong weight assignment, unclear wording, incomplete failure-chain description, or an implementation plan that needs clarification while preserving a plausible misconception.
REJECT: The candidate violates a core requirement. This label is appropriate if the belief is unnatural, the bug is not caused by the belief, the weight is forced, the candidate is not implementable, the scenario would require direct answer leakage, or the candidate is redundant.
Required review record. For every candidate, please complete the following fields: 1. root misconception in one sentence; 2. final decision: ACCEPT, REVISE, or REJECT; 3. accepted or corrected weight bucket; 4. pass/fail judgment for seed alignment, naturalness, causal consistency, implementation feasibility, tutoring suitability, and non-redundancy; 5. main reason for the decision; 6. required revision, if the decision is REVISE; 7. short reviewer note.
Please accept a candidate only when the learner belief, failure chain, and implementation plan form a coherent and educationally plausible scenario. Please reject a candidate only when it is unnatural, incoherent, unimplementable, non-diagnostic, redundant, or misaligned with the seed.

D Experimental Setup and API Configuration

This appendix reports the API routes, model identifiers, decoding parameters, and execution settings used in the construction and evaluation pipeline. We use two OpenAI-compatible API routes:

- **DS**: `https://dashscope.aliyuncs.com/compatible-mode/v1`
- **OR**: `https://openrouter.ai/api/v1`

For API-served models that do not expose immutable snapshot identifiers, we report the exact request model string, API route, call date range, and explicitly set request parameters as the reproducibility record. Unless otherwise stated, request parameters not listed in the tables were not explicitly set in the request body and were not treated as benchmark variables.

D.1 Misconception-Rule Construction

Misconception-rule construction uses multiple frontier models only to expand expert-written misconception seeds into task-conditioned misconception candidates. These calls were made between 2025-12-03 and 2026-01-02. The generation stage uses low-temperature sampling to preserve controlled diversity while keeping generated misconception plans aligned with the seed rule and task context. After generation, candidate screening, rule selection, and scenario realization were conducted by human experts according to the curation protocol in Appendix C.

Table 7: API configuration for misconception-rule generation.

Stage	Model string	Route	Temperature	Max tokens
Rule generation / expansion	`claude-opus-4-5`	OR	0.2	32000
Rule generation / expansion	`gemini-3-pro-preview`	OR	0.2	32000
Rule generation / expansion	`gpt-5.2`	OR	0.2	32000

The rule-generation models were used only to propose candidate misconception plans. They were not used to select accepted scenarios, implement buggy artifacts, assign diagnostic keys, or score formal teacher performance.

D.2 ZPD Filtering and Oracle Teachability Verification

CogTeachBench applies two pre-evaluation filtering stages. The first stage removes cases that the learner proxy can repair without teacher assistance. We use this as a zero-intervention ZPD filter: if a learner can independently repair the buggy artifact from the initial scenario context, the case is not retained as a stable instructional bottleneck.

Table 8: First-stage ZPD filtering configuration. This stage tests zero-intervention repairability.

Item	Setting
Learner model	`qwen2.5-7b-instruct`
API route	DS
Temperature	0.8
Top-p	0.9
Reflection max tokens	1024
Repair max tokens	4096
Purpose	Zero-intervention repair screening

The second stage verifies oracle teachability. Each retained candidate is paired with structured oracle guidance from a strong teacher model. The scenario is retained only if a lower-ability learner proxy can repair the case under this structured guidance. This stage removes cases that are too difficult or underspecified even when the relevant repair direction is supplied.

Table 9: Second-stage oracle-teachability verification configuration.

Role	Setting	Purpose
Oracle teacher model	`gpt-5.4`	Generates structured oracle guidance
Oracle teacher route	OR	OpenRouter API route
Oracle teacher temperature	0.4	Low-variance corrective guidance
Oracle teacher top-p	0.9	Controlled sampling
Oracle teacher max tokens	2048	Bounded oracle guidance length
Learner model	`qwen2.5-3b-instruct`	Lower-ability learner for teachability verification
Learner route	DS	DashScope API route
Learner temperature	0.1	Low-variance repair attempt
Learner top-p	0.85	Controlled decoding
Learner max tokens	4096	Post-guidance repair generation

D.3 Formal Teacher–Learner Evaluation

The formal benchmark evaluation uses the teacher models and learner profiles described in Section 4. Teacher models receive the problem statement, learner artifact, learner self-report, and verified reference solution, but not the misconception rationale or diagnostic key. Learner proxies are separated into an internal reasoning step and an external response step. This separation allows learner-profile controls, including acceptance behavior and overload-limited retained context, to be applied before the learner produces the next visible message.

Table 10: Formal teacher–learner interaction parameters.

Call type	Route	Temperature	Top-p	Max tokens
Teacher turn	OR / DS	0.1	–	–
Student brain	DS	0.2	0.9	256
Student mouth	DS	0.2	0.9	512
Post-test repair	DS	0.1	1.0	4096

The teacher-turn route depends on the evaluated teacher model: GPT-5.4, Gemini 3.1 Pro, and Claude Opus 4.6 use OR, while the Qwen-family teacher models use DS. The teacher-turn `top_p` and `max_tokens` fields were not explicitly set in the request body. The post-test stage uses a fixed `fix@3` protocol: each run allows three independent repair attempts, and the run is counted as successful only when at least one attempt passes the executable tests and no answer-transfer flag is raised.

D.4 Model Identifiers and API Routes

Table 11 reports the model identifiers and API routes used across the pipeline. Because several API-served systems expose only public aliases or preview names rather than immutable dated snapshots, we do not infer hidden snapshots. The exact request model string, API route, timestamped request/response logs, and decoding configuration serve as the reproducibility record.

Table 11: Model identifiers and API routes.

Role	Model string	Route	Call date range
Misconception construction	`claude-opus-4-5`	OR	2025-12-03–2026-01-02
Misconception construction	`gemini-3-pro-preview`	OR	2025-12-03–2026-01-02
Misconception construction	`gpt-5.2`	OR	2025-12-03–2026-01-02
Oracle teacher	`gpt-5.4`	OR	2026-01-01–2026-04-30
Formal teacher	`gpt-5.4`	OR	2026-01-01–2026-04-30
Formal teacher	`gemini-3.1-pro-preview`	OR	2026-01-01–2026-04-30
Formal teacher	`claude-opus-4-6`	OR	2026-01-01–2026-04-30
Formal teacher	`qwen3.5-plus`	DS	2026-01-01–2026-04-30
Formal teacher	`qwen3.5-flash`	DS	2026-01-01–2026-04-30
Learner proxy	`qwen2.5-3b-instruct`	DS	2026-01-01–2026-04-30
Learner proxy / self-teaching reference	`qwen2.5-7b-instruct`	DS	2026-01-01–2026-04-30

D.5 Cognitive-Load and Dialogue Judges

The β overload mechanism uses a bounded abstraction-level classifier as one component of the cognitive-load model. The abstraction classifier is intentionally constrained to a short structured output to reduce judge verbosity and improve parse reliability. For this classifier, provider-side thinking is explicitly disabled.

Table 12: Abstraction-level judge configuration for the cognitive-load model.

Item	Setting
Model	`qwen3-8b`
API route	DS
Temperature	0.0
Top-p	1.0
Max tokens	32
Maximum attempts	2
Extra body	`{"enable_thinking": false}`
Purpose	Bounded abstraction-level classification for the load model

Dialogue-level process scoring uses a structured LLM-as-judge protocol. The mainline script requests JSON-formatted outputs and uses low-temperature decoding. A direct-client fallback uses the client-side temperature setting shown in Table 13.

Table 13: Dialogue judge configuration for PSQ scoring.

Item	Setting
Model	`deepseek-v4-pro`
API route	DS
Mainline temperature	0.1
Mainline top-p	1.0
Mainline max tokens	768
Mainline max workers	20
Mainline response format	`{"type": "json_object"}`
Direct-client temperature	0.0
Purpose	Whole-dialogue PSQ dimension scoring

D.6 Logging and Reproducibility

All raw request and response payloads are cached with the scenario identifier, learner-profile identifier, prompt template identifier, API route, request model string, response metadata when available, timestamp, finish reason, and decoding configuration. We also log executable post-test results, repair-attempt outcomes, anti-spooning labels, cognitive-load traces, and dialogue-judge outputs. No teacher, learner, post-test, or judge response is manually edited before scoring.

E CBCS Calibration and Statistical Analysis

E.1 Bucket-Weight Calibration

CBCS first averages binary clearance within each misconception bucket and then applies calibrated bucket weights, so the final score is not driven by bucket-size imbalance. We calibrate these weights from the second-stage oracle teachability screen, after zero-intervention failures have already been filtered. For each bucket ω, let T_ω and U_ω denote the numbers of oracle-teachable and oracle-unteachable retained variants, respectively; teachability is decided by the hidden executable tests on the oracle-guided repair attempts. We define oracle unteachability odds $o_\omega = U_\omega/T_\omega$ and normalize by the shallowest bucket:

$$\lambda_\omega = \frac{o_\omega}{o_{w_1}}. \tag{5}$$

The resulting weights are $\lambda_{w_1} = 1.000$, $\lambda_{w_2} = 1.374$, and $\lambda_{w_3} = 2.155$. Thus, deeper misconception buckets receive larger CBCS weight because they are less often repairable even under structured oracle guidance.

Table 14: CBCS bucket-weight calibration from the oracle teachability screen.

Bucket	T_ω	U_ω	λ_ω
w_1	91	9	1.000
w_2	103	14	1.374
w_3	122	26	2.155

E.2 Stratified Bootstrap Confidence Intervals

We quantify CBCS and AZS uncertainty with a stratified nonparametric bootstrap over benchmark case IDs. For each teacher, we group the full-control cases by misconception bucket and sample case IDs with replacement within each bucket, preserving the original bucket counts. The sampled case IDs are then reused across all available full-control profiles for that teacher. CBCS is recomputed within each profile using the calibrated bucket weights and then averaged exactly as in Table 2. AZS is recomputed from the same bootstrap replicate as the signed 7B-minus-3B CBCS difference for each matched full-control α pair, then averaged across the two available pairs. We use 10,000 bootstrap replicates and report percentile 95% confidence intervals. This procedure estimates sampling uncertainty over misconception cases while preserving bucket composition, matched case IDs, and paired learner-profile structure.

Table 15: Stratified bootstrap CBCS and AZS confidence intervals for Table 2.

Teacher	Profiles	CBCS	CBCS CI	AZS CI
Gemini 3.1 Pro	4	0.581	[0.513, 0.650]	[-0.027, 0.123]
GPT-5.4	4	0.509	[0.444, 0.573]	[0.020, 0.200]
Claude Opus 4.6	4	0.486	[0.427, 0.544]	[-0.059, 0.122]
Qwen3.5 Plus	4	0.466	[0.405, 0.529]	[-0.060, 0.129]
Qwen3.5 Flash	4	0.313	[0.257, 0.370]	[-0.004, 0.149]
Qwen2.5-7B Instruct	2	0.062	[0.033, 0.093]	–

For the top-line ranking claim, we additionally use the same case-level bootstrap samples across teachers and compute paired differences. Gemini 3.1 Pro exceeds GPT-5.4 by 0.072 CBCS points with a 95% paired bootstrap interval of [0.014, 0.131]. The adjacent middle differences, GPT-5.4 over Claude Opus 4.6 (0.023; 95% CI [-0.028, 0.076]) and Claude Opus 4.6 over Qwen3.5 Plus (0.020; 95% CI [-0.032, 0.073]), are not statistically separated under this test. Qwen3.5 Plus exceeds Qwen3.5 Flash by 0.153 (95% CI [0.094, 0.214]). We therefore treat the strongest and weakest CBCS contrasts as stable, while interpreting the middle ordering descriptively. Paired comparisons for all teacher pairs are reported in Appendix E.4.

For AZS, GPT-5.4 is the only fully comparable teacher whose full-control interval is separated from zero (AZS $= 0.109$, 95% CI [0.020, 0.200]). The remaining intervals overlap zero, ruling out the simple monotonic interpretation that the 7B learner proxy reliably receives larger teaching gains than the matched 3B proxy. AZS should therefore be read as an ability-sensitivity diagnostic: positive values can occur, but they are teacher- and profile-conditioned rather than an automatic consequence of learner scale.

E.3 PSQ and PCE Confidence Intervals

We apply the same stratified case-level bootstrap to PSQ and PCE. In each of 10,000 replicates, task IDs are sampled with replacement within misconception buckets, and the sampled IDs are reused across all four full-control profiles of a teacher. PSQ and PCE are recomputed within each profile and then averaged exactly as in Table 2. We report percentile 95% confidence intervals. The intervals capture sampling uncertainty over misconception cases given the recorded judge outputs and the fitted load model.

Table 16: Stratified bootstrap PSQ and PCE confidence intervals for Table 2.

Teacher	PSQ [95% CI]	PCE [95% CI]
Gemini 3.1 Pro	0.779 [0.751, 0.804]	0.190 [0.158, 0.230]
GPT-5.4	0.793 [0.775, 0.812]	0.253 [0.212, 0.299]
Claude Opus 4.6	0.666 [0.642, 0.690]	0.112 [0.093, 0.133]
Qwen3.5 Plus	0.695 [0.675, 0.715]	0.084 [0.068, 0.102]
Qwen3.5 Flash	0.616 [0.590, 0.641]	0.057 [0.044, 0.072]

On both PSQ and PCE, the intervals of GPT-5.4 and Gemini 3.1 Pro lie above those of Claude Opus 4.6, Qwen3.5 Plus, and Qwen3.5 Flash. GPT-5.4 has the highest point estimate on both metrics, and Qwen3.5 Flash has the lowest.

E.4 Paired Between-Teacher Comparisons

Table 17 extends the paired CBCS comparison in Appendix E.2 to all teacher pairs and to PSQ and PCE.

Table 17: Paired between-teacher differences on full-control profiles (first minus second teacher in each pair), with 95% paired bootstrap intervals. *: the interval excludes zero.

Pair	ΔCBCS	ΔPSQ	ΔPCE
Gemini 3.1 Pro − GPT-5.4	$+0.072^*$ [+0.014, +0.131]	-0.015 [−0.042, +0.012]	-0.063^* [−0.092, −0.033]
Gemini 3.1 Pro − Claude Opus 4.6	$+0.096^*$ [+0.040, +0.151]	$+0.112^*$ [+0.083, +0.143]	$+0.079^*$ [+0.054, +0.107]
Gemini 3.1 Pro − Qwen3.5 Plus	$+0.115^*$ [+0.054, +0.177]	$+0.083^*$ [+0.054, +0.112]	$+0.106^*$ [+0.082, +0.134]
Gemini 3.1 Pro − Qwen3.5 Flash	$+0.268^*$ [+0.204, +0.335]	$+0.163^*$ [+0.130, +0.196]	$+0.133^*$ [+0.105, +0.166]
GPT-5.4 − Claude Opus 4.6	$+0.023$ [−0.028, +0.076]	$+0.127^*$ [+0.102, +0.153]	$+0.141^*$ [+0.109, +0.175]
GPT-5.4 − Qwen3.5 Plus	$+0.043$ [−0.011, +0.097]	$+0.098^*$ [+0.077, +0.120]	$+0.169^*$ [+0.136, +0.204]
GPT-5.4 − Qwen3.5 Flash	$+0.196^*$ [+0.139, +0.254]	$+0.178^*$ [+0.149, +0.206]	$+0.195^*$ [+0.160, +0.234]
Claude Opus 4.6 − Qwen3.5 Plus	$+0.020$ [−0.032, +0.073]	-0.029^* [−0.055, −0.003]	$+0.027^*$ [+0.013, +0.043]
Claude Opus 4.6 − Qwen3.5 Flash	$+0.173^*$ [+0.121, +0.224]	$+0.050^*$ [+0.024, +0.077]	$+0.054^*$ [+0.040, +0.070]
Qwen3.5 Plus − Qwen3.5 Flash	$+0.153^*$ [+0.094, +0.214]	$+0.080^*$ [+0.053, +0.107]	$+0.027^*$ [+0.013, +0.041]

For AZS, every paired between-teacher interval contains zero.

F Persona and Cognitive-Load Validity

Table 18: Six learner-profile conditions, each instantiated with 3B and 7B proxies (12 profiles).

	Baseline	High acceptance	Low acceptance	Overload only	High acceptance + overload	Low acceptance + overload
α	off	high	low	off	high	low
β	off	off	off	on	on	on

The standardized learner is implemented as a two-call student engine. The first call receives the current teacher message, previous dialogue, fixed misconception rationale, and active α/β state, and returns only an internal reaction. The second call receives that internal reaction and generates the visible student reply. The logged record keeps the visible reply, the internal reaction, the effective teacher message after any β intervention, and the overload policy metadata, allowing later audits of whether a reply came from full retention, partial retention, or continued misconception commitment.

We validate the learner-side parameters using two complementary analyses. First, we use a balanced learner-behavior audit with 2,400 dialogues: two teachers (GPT-5.4 and Qwen3.5 Flash), two ability tiers, six parameter profiles, and 100 cases per teacher–ability–profile cell. The 100 cases are sampled to preserve the misconception-bucket distribution of the full oracle-verified scenario pool. Each profile therefore contributes 400 dialogues. Binary behavioral markers are evaluated with two-sided Fisher exact tests: dialogue-level defensiveness for α, and turn-level confusion, request-slowdown,

and immediate code output for β. Second, we test whether β also changes the downstream teaching outcome in the formal teaching evaluation by comparing matched β-off and β-on CBCS under the same teacher, ability tier, α setting, and case IDs. For this outcome analysis, we use a paired stratified bootstrap over case IDs within misconception buckets with 10,000 replicates.

Parameter-disabled learners are not a substitute baseline. Without explicit learner parameters, the student proxy does not naturally exhibit the target overload behavior. In the balanced audit, `params_off` produces 400 dialogues and 1,698 student turns, but only 0.65% of turns express confusion, 0.47% request slowdown or restatement, and no dialogue triggers overload metadata. At the same time, 30.09% of turns contain immediate code output. Thus, disabling α/β does not yield an implicit learner that already covers the intended acceptance and partial-retention states; it yields a different behavioral regime.

α changes acceptance behavior. The α parameter is intended to control how readily the learner accepts teacher guidance. The behavioral audit supports this manipulation. When β-on and β-off profiles are pooled, α-low learners produce defensive dialogues more often than α-high learners (58.25% vs. 50.25%; Fisher $p = 0.00156$), and require longer interactions on average (4.598 vs. 4.374 student turns per dialogue). The same direction appears without β overload (53.75% vs. 47.50%), and becomes clearer in the full-control β-on setting (62.75% vs. 53.00%; Fisher $p = 0.00647$). This confirms that α is not merely a label: it changes how much resistance and evidence demand the teacher faces.

Table 19: α validity in the balanced learner-behavior audit. Defensive rates are dialogue-level; turns/dialogue is descriptive.

Comparison	α-high	α-low	Test
All β settings: defensive dialogue rate	50.25%	58.25%	Fisher $p = 0.00156$
All β settings: student turns/dialogue	4.374	4.598	–
β off: defensive dialogue rate	47.50%	53.75%	–
β on: defensive dialogue rate	53.00%	62.75%	Fisher $p = 0.00647$
β on: student turns/dialogue	4.708	5.100	–

β changes overload behavior. The β parameter is intended to model partial retention under cognitively overloaded teacher turns, not to globally weaken the learner. This distinction is visible in the logged behavior. Relative to `params_off` turns, overloaded β-on turns show substantially more confusion and slowdown requests, and substantially less immediate code output (Table 20). All Fisher tests against `params_off` are significant. The weakest beta-only condition, α-off β-on, already changes behavior; adding α profiles strengthens the overload symptoms, especially confusion and slowdown requests.

Table 20: β validity on overloaded turns. Rates are turn-level; Fisher tests compare β-on overload columns against `params_off`.

Marker	`off`	β only	α-high+β	α-low+β
Confused	0.65%	9.52%***	29.10%***	25.35%***
Request slowdown	0.47%	2.84%***	19.66%***	13.90%***
Immediate code output	30.09%	15.82%***	4.59%***	11.84%***

*** Two-sided Fisher exact test against `params_off`: confused $p \leq 1.91 \times 10^{-12}$, request slowdown $p \leq 1.79 \times 10^{-6}$, and immediate code output $p \leq 2.68 \times 10^{-12}$.

β also shifts measured teaching gain. The behavioral changes above are not isolated prompt artifacts; they also affect the outcome metric. Across the three teachers with complete 12-profile coverage, β-on lowers CBCS relative to matched β-off profiles by 0.070 on average, with a paired stratified bootstrap 95% CI of [0.035, 0.106] and no sign reversals in 10,000 bootstrap replicates ($p < 2 \times 10^{-4}$). Individual teacher–α cells vary: the effect is statistically separated from zero for Qwen3.5

Flash under α-high and Qwen3.5 Plus under α-high, while GPT-5.4 shows smaller directional drops whose intervals include zero. This pattern is consistent with the main-text interpretation that stronger pacing and more localized instruction can reduce, but not remove, sensitivity to learner-retention limits.

Table 21: Matched CBCS drop from β-off to β-on. Drops are β-off minus β-on, averaged over matched 3B/7B pairs; intervals use paired stratified bootstrap.

Teacher	α-high Δ	α-low Δ	Test
GPT-5.4	0.057 [-0.024, 0.138]	0.039 [-0.032, 0.111]	n.s.
Qwen3.5 Flash	0.116 [0.041, 0.193]	0.068 [-0.002, 0.139]	high: $p = 0.003$
Qwen3.5 Plus	0.100 [0.027, 0.174]	0.042 [-0.045, 0.129]	high: $p = 0.006$
Aggregate	0.070 [0.035, 0.106]		$p < 2 \times 10^{-4}$

G Beta Calibration Details

The β intervention uses three implementation signals. F_{inst} captures instruction demand from the teacher turn, including the number and intensity of requested learning actions. F_{formal} captures formal complexity from code, mathematical notation, control-flow structure, and dense technical language. F_{abstract} captures the abstraction level of the explanation through a bounded classification judge. These features are used only to decide how much of an overloaded teacher turn the learner can retain; they do not directly score teacher quality. The abstraction classifier is selected by agreement with three-expert annotations in Appendix I.1.

G.1 Student Subjective-Load Rating Guideline

The coefficients and overload thresholds in the β load model are calibrated from student subjective-load ratings rather than from LLM-generated load labels. We collect ratings for 557 teaching segments. Each rating item presents a programming task context, the learner's current buggy attempt or misconception state, the recent dialogue context when available, and one target teacher turn. Students are asked to judge how cognitively demanding the target teacher turn would be for them to understand and use for repairing the bug. The ratings come from 13 student volunteers in two groups that differ in relative learning ability: a stronger group of 7 and a weaker group of 6. Every volunteer independently rates all 557 segments. The continuous load target of each segment is the median of the 13 ratings on the 0–10 scale. Overload labels are aggregated separately within each group by majority vote over the binary overload judgments. Because the weaker group has an even number of raters, 19 segments (3.4%) received a 3:3 tie; for these, a seventh rater made the final decision. The resulting overload labels are positive for 219 segments in the weaker group and 32 segments in the stronger group.

Student Guideline for Subjective Cognitive-Load Rating

Rating task. Thank you for rating the cognitive load of programming tutoring messages. For each item, please read the task context, the student's current buggy attempt or misconception state, the recent dialogue context if provided, and the target teacher message. Your goal is to judge how mentally demanding the target teacher message would feel if you were the student trying to understand it and use it to repair the bug.
What to rate. Please rate the cognitive load imposed by the target teacher message. Focus on how much information you would need to hold in mind, connect, and apply. Do not rate whether the teacher is polite, fluent, friendly, or generally correct. Do not rate whether you like the style of the explanation. The rating should reflect perceived processing burden.
Use of context. Please use the task context and dialogue context only to understand what the teacher message is asking the student to process. If a message is easy in isolation but difficult because it refers to several previous ideas, rate the overall burden in context. If the message is long but mostly repeats a single clear point, do not rate it as high load only because it is long.
Rating scale. Please assign one integer score from 0 to 10.
0: None. Nothing to process beyond reading the message.
1: Minimal. The message asks for one simple check or local edit.



Figure 3: Participant-facing questionnaire interface for student subjective cognitive-load calibration. The item shows the programming task context, learner misconception state, target teacher message, and a 0–10 perceived-load rating scale.

2: Very low. The message is concrete and local, with only a little terminology.
3: Low. The message requires one local reasoning step, such as a boundary case or a simple trace.
4: Mild. The message requires one reasoning step plus a code change to apply.
5: Moderate. The message requires coordinating several local details, such as a trace plus an update rule.
6: Elevated. The message contains dense reasoning or several linked steps.
7: High. The message introduces nontrivial notation or a broader concept that must be connected to the current code.
8: Very high. The message requires holding multiple ideas in mind at once, such as a strategy change, an invariant, and a trace.
9: Severe. The message is hard to retain because it combines many demands in one turn.
10: Overwhelming. The message contains too many demands to retain or use.
Specific cues. When rating, please consider: 1. the number of separate actions the teacher asks the student to perform; 2. the amount of code, notation, or formal reasoning in the message; 3. the abstraction level of the explanation; 4. whether the message combines several repair goals in one turn; 5. whether the student would need to remember earlier dialogue content to make sense of the message.
Important distinction. A message can be high quality and still impose high cognitive load. A message can also be low quality but low load. Please rate load only, not teaching quality.
Output. For each item, please provide: 1. one integer load score from 0 to 10; 2. an optional short note explaining the main reason for the score.

G.2 Load-Model Fitting

The composite load score is fitted as

$$L = 0.3000F_{\text{inst}} + 0.4800F_{\text{formal}} + 0.2200F_{\text{abstract}}. \tag{6}$$

The target variable is the median 0–10 subjective-load rating of the 13 student raters for each teaching segment. The coefficients are estimated from the 557 teaching segments annotated through the student questionnaire in Appendix G.1. The feature-level scoring parameters in Table 23, including the four scaffolding credits, are specified by educational experts. The three top-level weights are then selected from 1,326 nonnegative simplex candidates summing to one (grid step 0.02). Each candidate is evaluated by five-fold cross-validation with folds grouped by the 113 source problems, and the candidate with the highest mean held-out Spearman correlation is selected; the selected weights are 0.30, 0.48, and 0.22. The fit achieves full-sample Spearman $\rho = 0.681688$ and Pearson $r = 0.657013$, with 5-fold grouped cross-validation mean Spearman $= 0.650186$ (± 0.082988). Measured on the 0–10 rating scale, i.e., between $10L$ and the median rating, the RMSE is 2.504 (equivalently 0.2504 on a $[0, 1]$-normalized scale). The predicted L range spans $[0.058, 0.974]$.

The operational overload thresholds are calibrated separately against the group-level overload labels: the weaker group's labels (219 positive of 557) calibrate the 3B learner, and the stronger group's labels (32 positive) calibrate the 7B learner. For each group, every distinct value of L (rounded to four decimals) is a candidate threshold t, a segment is predicted as overloaded when $L \geq t$, and we select the t that maximizes balanced accuracy, which is equivalent to maximizing Youden's J. Table 22 reports the result. The 3B learner is assigned $L_{3\mathrm{B}} = 0.6847$ and the higher-ability 7B learner $L_{7\mathrm{B}} = 0.7868$, so the 7B learner can retain a higher-load teacher turn before overload is triggered.

Table 22: Overload-threshold calibration against group-level student overload labels. Thresholds maximize balanced accuracy (Youden's J) under the rule $L \geq t$.

Rater group → learner	Threshold	Balanced acc.	AUC
Weaker (219/557 overload) → 3B	0.6847	0.7834	0.851
Stronger (32/557 overload) → 7B	0.7868	0.8668	0.894

When an overload intervention triggers, the learner receives a sentence-level truncated view of the current teacher message that retains approximately the head 25% and tail 25% while dropping the middle segment. This head–tail retention rule is specified with educational experts so that limited retention is reproducible across teachers and runs.

Table 23: Runtime cognitive-load features and β-on simulation constants. Feature-level constants are specified by educational experts; the top-level weights are fitted as described in Appendix G.2.

Feature	Raw computation	Normalization and discounts
F_{inst}	Sentence action/root lemmas are mapped to Bloom-style demand groups: recall/check/find/verify verbs use 1.0, apply/trace/compute/explain verbs use 1.35, create/prove/rewrite/synthesize verbs use 1.8, auxiliary control verbs use 0.2, and other actionable verbs use 0.65. Repeated action demands are softened after the third action by a decay rate of 0.85.	Adjacent coherent steps receive a 0.4 discount per pair, capped at 0.85. A pair is coherent if embedding similarity is at least 0.65 and either noun-chunk overlap is at least 0.4 or code-slot continuity holds within the same action family; repeated question templates also qualify when template similarity is at least 0.92. Strict response constraints add 0.18 per hit, capped at 0.8. Abstract-demand sentences without local bridge support add 0.28 per gap, capped at 0.9. The final raw score is capped at 3.8 and divided by 3.8.
F_{formal}	Code burden is scored from Tree-sitter function, control-flow, comprehension, nesting, call-chain, error, missing-node, and statement features, with Radon/fallback complexity when parsing is limited. Formula burden counts operators such as `\sum`/`\prod` (0.45), `\frac` (0.30), max/min (0.22), argmax/argmin (0.32), subscripts (0.08), and superscripts (0.18). Text traces add dependency-distance load above 3.5, transition/comparison/step/literal/chain cues, active state variables, and mental-operation verbs.	Teacher quotes of the learner's own code or formulas are included with a 0.1 self-reference scale. Explicit localization, replacement, decomposition, and examples subtract scaffolding credit with weights 0.14, 0.24, 0.08, and 0.14, capped by both 0.85 and 40% of the base formal load. The normalized value is $x/(x+1)$.
F_{abstract}	A low-temperature `F_abstract` classifier labels the teacher turn as `LEVEL_1`, `LEVEL_2`, or `LEVEL_3`, mapped to 0.1, 0.4, and 1.0.	The base abstraction score is multiplied by $\text{clip}(1+0.12p-0.18s, 0.72, 1.18)$, where p is bounded response-pressure strength and s is bounded support strength from decomposition, localization, replacement, examples, bridges, and worked-example sentences.

Thus the instruction component is not a raw count of questions. It first extracts action demands, then discounts adjacent questions or steps that continue the same semantic operation. Conversely, terse high-pressure prompts can still add load through response-constraint hits, and abstract requests without bridge sentences add a scaffolding-gap penalty. The formal component separates teacher-provided new burden from references to the learner's own code; quoting the learner's code is not free, but receives the 0.1 self-reference scale because it is less novel than an introduced code fragment. The abstraction component is also delivery-sensitive: an abstract explanation with explicit localization and examples is scored lower than an equally abstract directive that asks for an immediate answer.

Algorithm G.1: Cognitive-Load Scoring and Overload Intervention

Input: teacher turn m_t, learner persona p, task state x, dialogue history H
Output: effective teacher message $\tilde{m}_t$, load record z_t

$v_t \leftarrow \mathrm{TeacherGuidanceView}(m_t)$
$S \leftarrow \mathrm{SentenceSplit}(v_t)$; $r_{\mathrm{inst}} \leftarrow 0$; $d_{\mathrm{coh}} \leftarrow 0$
for each non-support sentence $s_j \in S$ **do**
  $(\ell_j, a_j) \leftarrow \mathrm{ActionLemmaAndBloomWeight}(s_j)$
  $r_{\mathrm{inst}} \leftarrow r_{\mathrm{inst}} + \mathrm{ActionDecay}(a_j, \mathrm{count}(\ell_j))$
end for
for each adjacent pair (s_j, s_{j+1}) **do**
  if $\mathrm{SemanticOrTemplateContinuity}(s_j, s_{j+1})$ **then** $d_{\mathrm{coh}} \leftarrow d_{\mathrm{coh}} + 0.4$
end for
$F_{\mathrm{inst}} \leftarrow \mathrm{clip}((\max\{0, r_{\mathrm{inst}} - \min(d_{\mathrm{coh}}, 0.85)\} + r_{\mathrm{constraint}} + r_{\mathrm{gap}})/3.8, 0, 1)$
$r_{\mathrm{formal}} \leftarrow \mathrm{CodeLoad}(m_t) + \mathrm{FormulaLoad}(m_t) + \mathrm{TraceLoad}(m_t) - \mathrm{ScaffoldCredit}(m_t)$
$F_{\mathrm{formal}} \leftarrow \max\{0, r_{\mathrm{formal}}\}/(\max\{0, r_{\mathrm{formal}}\} + 1)$
$b_{\mathrm{abs}} \leftarrow \mathrm{AbstractJudge}(m_t) \in \{0.1, 0.4, 1.0\}$
$F_{\mathrm{abstract}} \leftarrow \mathrm{clip}(b_{\mathrm{abs}} \cdot \mathrm{clip}(1 + 0.12p_t - 0.18s_t, 0.72, 1.18), 0, 1)$
$L_t \leftarrow 0.3000F_{\mathrm{inst}} + 0.4800F_{\mathrm{formal}} + 0.2200F_{\mathrm{abstract}}$
$\tau_p \leftarrow 0.6847$ for 3B learners and 0.7868 for 7B learners
if $L_t \geq \tau_p$ **then**
  $\tilde{m}_t \leftarrow \mathrm{HeadTailTruncate}(m_t, \mathrm{head} = 25\%, \mathrm{tail} = 25\%)$ with an overload marker
else $\tilde{m}_t \leftarrow m_t$
$z_t \leftarrow (F_{\mathrm{inst}}, F_{\mathrm{formal}}, F_{\mathrm{abstract}}, L_t, \tau_p, \tilde{m}_t)$

G.3 Scaffolding-Credit Sensitivity

The four scaffolding credits in F_{formal} (localization 0.14, replacement 0.24, examples 0.14, and decomposition 0.08) are fixed constants of the load model. To test whether the reported load and PCE comparisons depend on their exact values, we recompute turn-level load, overload decisions, and run-level PCE on the recorded teacher trajectories under four alternative settings: no credit, 75% and 125% of the reported credits, and a uniform credit of 0.15. The analysis covers 14,918 teacher turns from the 23 mainline β-on runs whose turn-level load records are available. Dialogues and post-test outcomes are held fixed, so the analysis isolates the effect of the credits on load scoring and overload decisions.

Table 24: Fixed-trajectory sensitivity of load, overload decisions, and PCE to the scaffolding credits. PCE changes are relative to the reported configuration.

Credit setting	Overload agreement	Load ρ	Mean PCE change	Max run change	Inversions
No credit	96.00%	0.99348	−4.04%	5.19%	0
75% of reported	99.48%	0.99961	−0.54%	0.99%	0
Reported	100%	1.00000	0	0	0
125% of reported	99.78%	0.99977	+0.31%	0.70%	0
Uniform 0.15	99.77%	0.99971	+0.23%	0.42%	0

The uniform setting removes the difference between the replacement and decomposition credits, yet changes only 35 of 14,918 overload decisions. Under every setting, the 23 run-level PCE scores keep the same order (Spearman 1.0, zero pairwise inversions), and the common-profile teacher ranking by PCE is unchanged. Removing all credits lowers mean PCE by 4.04% but still preserves the full ranking. The comparative PCE results therefore do not depend on the specific credit values.

H Anti-Spooning Validity Checks

Anti-spooning validity is evaluated with two complementary checks: agreement with expert adjudication on sampled teacher turns, and separation between a white-box oracle-teacher control and formal benchmark runs. This check is part of benchmark-integrity control. Because an LLM

learner can reproduce teacher-provided code from its context, a teacher that restates the answer would otherwise produce spurious post-test gains; the check prevents such answer transfer from inflating CBCS, rather than contributing to the PSQ process rubric. The detector is deliberately scoped to answer-level code transfer. It does not treat natural-language explanation, mathematical derivation, or conceptual guidance as answer transfer unless the teacher output contains executable or structurally solution-equivalent Python code, because such guidance still requires the learner proxy to translate the instruction into a sandbox-passing repair.

Detector implementation. The runtime first extracts candidate Python snippets from the teacher response, prioritizing fenced `python`/`py` blocks, unlabeled fenced blocks that pass Python-like checks, and whole-message code when the response is itself code-like. Placeholder scaffolds are whitelisted before any answer-transfer decision: examples include `pass`, ellipses, `NotImplementedError`, TODO/FIXME-style comments, and dangling placeholder identifiers. The remaining snippets are checked by two complementary mechanisms. The executable direct-answer veto asks whether a non-whitelisted snippet is at least half the length of the reference solution and passes the benchmark tests. Such snippets are first run on one test case; only snippets that pass this fast check are run on the full test set. Passing all tests triggers `sample_vetoed` and the record receives no CBCS credit.

The structural-transfer detector is based on Tree-sitter Python. It first extracts a severity-specific anchor from the reference solution and buggy code, then compares the teacher snippet against that anchor. Let

$$\mathrm{sim} = 0.55\,\mathrm{sim}_{\mathrm{seq}} + 0.25\,\mathrm{sim}_{\mathrm{hist}} + 0.20\,\mathrm{sim}_{\mathrm{shape}}, \tag{7}$$

where the terms compare normalized node sequences, node-type histograms, and tree-shape features. The decision score is $\mathrm{joint} = \mathrm{sim} \times \mathrm{coverage}$, where coverage is the fraction of anchor named nodes covered by the best matched subtree. Exact quotations of the learner's buggy code or previous learner code are exempted. A structural-transfer match does not stop the dialogue online, but the final evaluator treats any direct-answer or structural-transfer flag as answer transfer, setting the outcome indicator R_i to zero.

Table 25: Structural-transfer anchors by misconception bucket. Anchors define when teacher code is too solution-specific.

Bucket	Anchor	Extraction	Threshold
w_1	Repair patch	Parse the buggy code and reference solution, align semantic line differences, and select the smallest patch candidate covering the changed solution lines. Leaf or operator-level changes are promoted to a viable assignment, return, condition, call, or subscript-level node when needed.	$\mathrm{sim} \geq 0.90$, $\mathrm{joint} \geq 0.55$; minimum one line and three named nodes.
w_2	Repair mechanism	Start from diff-supported solution nodes and climb to the nearest mechanism node, such as an `if`, `for`, `while`, `with`, `try`, `match`, function, or block. If the mechanism body is broad, derive a focus window around statements that contain diff nodes or intersect changed lines.	$\mathrm{sim} \geq 0.70$, $\mathrm{joint} \geq 0.60$; minimum three lines and ten named nodes.
w_3	Solution core	Promote diff-supported solution nodes and score ancestor candidates among core node types; if no diff pair exists, scan the solution tree. The score favors control flow, statements, calls, depth, named nodes, and line count, while penalizing I/O wrappers, generic `main`/`solve` functions, and exception wrappers.	$\mathrm{sim} \geq 0.72$, $\mathrm{joint} \geq 0.62$; minimum three lines and ten named nodes.

Algorithm H.1: Anti-Spooning Check for One Teacher Turn

Input: teacher message m_t, buggy code b, reference solution $s^\star$, tests T, bucket w
Output: direct-answer veto v_{exec}, structural-transfer flag v_{struct}, evidence record e_t

$C \leftarrow \text{ExtractPythonSnippets}(m_t)$ from fenced blocks or code-like whole messages
$v_{\text{exec}} \leftarrow \text{false}; \quad v_{\text{struct}} \leftarrow \text{false}; \quad e_t \leftarrow \emptyset$
for each snippet $q \in C$ **do**
  if $\text{PlaceholderWhitelist}(q)$ **then continue**
  if $|q| \geq 0.5|s^\star|$ and $\text{PassesFirstTest}(q, T)$ **then**
    $v_{\text{exec}} \leftarrow \text{PassesAllTests}(q, T)$; **if** v_{exec} **then** record direct-answer evidence
  end if
  if $\text{QuotesStudentOwnedCode}(q, b, H)$ **then continue**
  $A_w \leftarrow \text{ExtractAnchor}(b, s^\star, w)$ using Table 25
  if $A_w = \emptyset$ or A_w is too small for bucket w **then continue**
  $M \leftarrow \text{BestTreeSitterSubtreeMatch}(q, A_w)$
  $\text{sim} \leftarrow 0.55\text{sim}_{\text{seq}}(M, A_w) + 0.25\text{sim}_{\text{hist}}(M, A_w) + 0.20\text{sim}_{\text{shape}}(M, A_w)$
  $\text{joint} \leftarrow \text{sim} \cdot \text{coverage}(M, A_w)$
  if sim and joint exceed the threshold for w **then** $v_{\text{struct}} \leftarrow \text{true}$
  if $w \in \{w_2, w_3\}$ and $\text{ShortPatchEvent}(q, A_w)$ **then** record covered anchor nodes
end for
if deduplicated short-patch coverage for w_2/w_3 exceeds 0.70 **then** $v_{\text{struct}} \leftarrow \text{true}$
return $(v_{\text{exec}}, v_{\text{struct}}, e_t)$

Short-patch accumulation is enabled only for w_2/w_3, because a teacher can leak a mechanism or solution core across several small snippets without any single snippet exceeding the full structural threshold. Each recorded patch must have structural similarity at least 0.90, token similarity at least 0.95, at most two nonempty lines, and 3–40 named nodes; after deduplication, cumulative anchor coverage must exceed 0.70 to produce an episode-level structural-transfer flag. Snippets or anchors below the configured line/node minima do not produce binary structural-transfer matches, which avoids penalizing harmless fragments such as a single variable name or placeholder expression.

Reference-conditioned coverage. The verified reference solution is part of the teacher context. Because teacher generation is conditioned on this solution, code that transfers the answer tends to follow reference-aligned structures rather than an independently designed equivalent implementation, which is the form the anchor-based structural detector targets. Complete solutions are handled independently of code layout by the executable veto: any non-whitelisted snippet at least half the length of the reference solution that passes all hidden tests is flagged. Tree-sitter matching complements this veto for incomplete, solution-specific transfer through bucket-specific anchors, best-subtree matching, joint coverage, and cumulative short-patch detection.

Parameter selection and expert agreement. The detector separates expert-specified semantic rules from numerically selected parameters. Educational experts specify the rule-level constraints, such as requiring teacher-provided code to pass every hidden test before the executable veto applies. The structural mixture weights w_{seq}, w_{hist}, and w_{shape}, with $w_i \in [0, 1]$ and $\sum_i w_i = 1$, are grid-searched together with the per-bucket similarity and joint-score thresholds, and the configuration with the highest positive-class F_1 against expert labels is selected. This yields the weights 0.55/0.25/0.20 and the thresholds in Table 25; for w_3, where candidate thresholds were nearly tied on a 0.05 grid, the range $[0.5, 0.8]$ is refined at 0.01 resolution. The expert labels cover 150 teacher turns spanning 36 cases and six teacher models: 75 turns drawn at random from approximately 3,000 teacher outputs, of which 2 were labeled as answer transfer, and 75 turns sampled at random from 123 suspected transfers identified by LLM-assisted triage over the same pool and not caught by the executable veto, of which 46 were labeled as answer transfer. Experts labeled all turns blind to detector outputs. Each turn is assigned a binary expert label indicating whether it contains answer transfer, and the detector prediction is the union of executable direct-answer veto and structural-transfer flags. Table 26 reports the resulting confusion matrix. Against expert adjudication, the detector achieves 89.3% accuracy (134/150; Wilson 95% CI [83.4%, 93.3%]), 82.0% precision for answer-transfer flags (41/50; [69.2%, 90.2%]), 85.4% recall (41/48; [72.8%, 92.8%]), 91.2% specificity (93/102; [84.1%, 95.3%]), $F_1 = 0.837$, and Cohen's $\kappa = 0.758$. The residual errors are balanced between false positives and false negatives (9 vs. 7), indicating that the detector is not simply labeling all code-bearing guidance as cheating or, conversely, missing most answer transfer.

Table 26: Anti-spooning detector confusion matrix against expert labels. Positive means answer transfer: executable leakage or structural transfer.

	Expert adjudication	
Detector	Transfer	No transfer
Answer-transfer flag	41	9
No answer-transfer flag	7	93

Oracle/formal separation. Under the structural-transfer detector, the oracle-teacher outputs trigger 225/365 times (61.64%, Wilson 95% CI [56.56%, 66.49%]). In formal benchmark records, any answer-transfer label appears in 75/3132 records (2.39%, Wilson 95% CI [1.91%, 2.99%]); structural transfer alone appears in 73/3132 records (2.33%), and direct-answer veto appears in 2/3132 records (0.064%). The oracle/formal separation is significant under Fisher's exact test ($p = 1.51 \times 10^{-12}$), and all formal records flagged for answer transfer contain code blocks, compared with 51.82% of unflagged records ($p = 6.00 \times 10^{-13}$). In the strict mainline subset, flagged records have higher raw post-test success than unflagged records (58.82% vs. 41.13%; Fisher $p = 0.0509$), suggesting that the detector identifies answer-transfer behavior that can inflate uncorrected outcomes.

I LLM-AS-JUDGE CONSISTENCY PROTOCOL

CogTeachBench uses LLM components in bounded roles: seed-conditioned misconception-candidate expansion, abstraction-level classification for F_{abstract}, and whole-dialogue process scoring for PSQ. Candidate acceptance remains expert-curated as described in Appendix C.2. For PSQ, the consistency check is dimension-wise because RCA, SGR, ETC, and MSS measure different pedagogical constructs and can fail independently. For the β load model, the abstraction-level classifier is checked as a bounded categorical judgment rather than as an open-ended critique. The main paper reports only metrics for which failed or unparsable judge calls are absent or negligible in the evaluated runs; full prompt text and rerun metadata are kept with the released artifacts.

I.1 ABSTRACTION-LEVEL CLASSIFIER AGREEMENT

The abstraction component F_{abstract} is computed for 59,793 teacher turns. To validate the runtime abstraction-level classifier, we sample 600 teacher turns and ask three human experts to independently annotate each turn as LEVEL 1, LEVEL 2, or LEVEL 3. The human reference label is obtained by majority vote among the three experts. We then compare candidate judge models against this human reference and select the model with the highest agreement for runtime use. Agreement is measured as exact label agreement with the three-expert majority label. For the human row, agreement denotes the average pairwise agreement among the three human experts.

Table 27: Abstraction-level judge agreement with three-expert majority labels on 600 teacher turns.

Judge model	Agreement
Qwen3-4B	78.5%
Qwen3-8B	**85.3%**
Llama-3.1-8B-Instruct	81.7%
Gemma-3-12B-IT	80.9%
DeepSeek-R1-Distill-Qwen-7B	82.4%
Human inter-annotator agreement	89.1%

Qwen3-8B achieves the highest agreement with the three-expert reference among the evaluated judge models. We therefore use Qwen3-8B as the runtime classifier for F_{abstract} over all 59,793 teacher turns.

I.2 Human Annotation Guideline for Abstraction Levels

The following guideline was provided to human experts when annotating abstraction levels for the 600 sampled teacher turns.

Human Annotation Guideline for F_{abstract}

Annotation goal. Thank you for annotating the abstraction level of teacher guidance messages in programming tutoring dialogues. Each item contains one teacher message. Please judge only the abstraction level of the current teacher message, not whether the message is pedagogically good, correct, polite, or effective.
Available labels. Please assign exactly one of the following labels.
LEVEL 1: OPERATIONAL GUIDANCE. Use this label when the teacher mainly gives concrete implementation guidance. Typical cases include line-level edits, exact variable updates, direct replacement instructions, explicit code changes, or highly local instructions about what to change in the learner's program. The learner can usually act on the message by modifying a small part of the code without reorganizing the solution logic.
LEVEL 2: TACTICAL OR LOCAL REASONING GUIDANCE. Use this label when the teacher explains a local reasoning mechanism without directly giving the final code. Typical cases include control-flow reasoning, boundary behavior, loop termination, branch coverage, state transition, local invariant checks, concrete counterexamples, or dry runs over a small input. The learner usually needs to reason about how a local part of the program behaves before making a repair.
LEVEL 3: STRATEGIC OR HIGH-LEVEL GUIDANCE. Use this label when the teacher introduces or relies on a broader algorithmic idea. Typical cases include an algorithmic paradigm, data structure choice, recurrence formulation, global invariant, proof idea, state representation, problem reformulation, or substantially different solution strategy. The learner usually needs to reorganize the solution logic or reinterpret the task at a higher level.
Decision rule for mixed messages. If a teacher message contains multiple levels, please choose the level that imposes the highest working-memory demand on the learner. For example, if a message gives one concrete edit but also introduces a new global invariant or algorithmic paradigm, assign LEVEL 3. If a message gives a local code edit and also explains a boundary trace, assign LEVEL 2 unless the main burden is a broader strategy change.
What not to judge. Please do not judge whether the teacher is correct, whether the learner eventually succeeds, whether the explanation is fluent, or whether the teacher should have used a different strategy. The label should reflect only the abstraction level of the current teacher message.
Context use. Please annotate the message as it appears. If task context is available, you may use it only to understand whether the message is local, tactical, or strategic. Do not infer hidden intent from the rest of the dialogue.
Output format. For each item, please output exactly one label: LEVEL 1, LEVEL 2, or LEVEL 3. If the message is ambiguous, choose the highest level that is clearly present and briefly note the ambiguity in the comment field.

I.3 PSQ Judge Model Selection and Agreement

PSQ is scored at the whole-dialogue level over four dimensions: root-cause concept attribution (RCA), Socratic guidance ratio (SGR), execution trace and cognitive conflict (ETC), and micro-scaffolding slice (MSS). To select the runtime PSQ judge, we sample 150 dialogue records and ask three human experts to independently score all four PSQ dimensions for each record using the same rubric. The sample is stratified by teacher model, learner profile, post-test success status, and preliminary PSQ score range. The human reference score for each dimension is obtained by the median of the three expert scores. We then compare candidate judge models against this human reference and select the model with the highest agreement for runtime use. Agreement is measured at the dimension-record level: a judge score is counted as agreeing with the human reference if it differs from the human median by at most one point. For the human row, agreement denotes the average pairwise within-one agreement among the three human experts.

Table 28: PSQ judge agreement with three-expert reference scores on 150 dialogues.

Judge model	Agreement
GPT-5.2	83.6%
Gemini 3 Pro Preview	82.9%
Claude Opus 4.5	82.1%
Qwen3.5 Plus	78.4%
DeepSeek-V4-Pro	**87.6%**
Human inter-annotator agreement	91.3%

DeepSeek-V4-Pro achieves the highest within-one agreement with the three-expert reference among the evaluated PSQ judge models. We therefore use DeepSeek-V4-Pro as the runtime dialogue judge for PSQ scoring over the full evaluation set. The runtime PSQ judge prompt is provided in Appendix L.4.

I.4 Human Annotation Guideline for PSQ

The following guideline was provided to human experts when annotating PSQ scores for sampled dialogue records.

Human Annotation Guideline for PSQ

Annotation goal. Thank you for scoring the process-level scaffolding quality of programming tutoring dialogues. Each item contains the task context, the learner's initial query, the buggy code, the diagnostic key, the full tutoring dialogue, and teacher-turn cognitive-load annotations. Please score the quality of the tutoring process, not the general fluency or politeness of the teacher.
General scoring rule. Please score each dimension independently. Use the full dialogue as evidence.
Scale. Use a 0–4 integer score for each dimension.
0: The dimension is absent or actively violated.
1: The dimension is weakly present, mostly superficial, or poorly matched to the learner state.
2: The dimension is partially present but incomplete, inconsistent, or only locally useful.
3: The dimension is clearly present and mostly effective, with minor omissions.
4: The dimension is strong, explicit, well grounded in the dialogue evidence, and well matched to the learner state.
Dimension 1: Root-cause concept attribution (RCA). Score whether the teacher identifies and works on the causal misconception or invariant behind the failure. High scores require the teacher to move beyond surface edits and target the underlying concept that explains why the learner's code fails. Low scores should be assigned when the teacher mainly points to local code changes, gives a patch, or leaves the underlying concept unnamed.
Dimension 2: Socratic guidance ratio (SGR). Score whether the teacher keeps the learner cognitively active through questions, checkpoints, reflective prompts, or guided discovery. High scores require meaningful learner involvement rather than rhetorical or empty questions. Low scores should be assigned when the teacher mainly gives direct fixes, formulas, near-complete code, or questions that do not help the learner reason.
Dimension 3: Execution trace and cognitive conflict (ETC). Score whether the teacher uses concrete evidence to resolve visible confusion or resistance. Useful evidence includes counterexamples, edge cases, dry runs, variable traces, contradiction with the learner's belief, or explicit comparison between expected and actual behavior. Low scores should be assigned when the teacher repeats abstract explanations while the learner remains confused, or when the dialogue lacks concrete evidence despite clear need.
Dimension 4: Micro-scaffolding slice (MSS). Score whether the teacher decomposes the repair into small, local, single-focus steps matched to the learner state. High scores require the teacher to sequence guidance so that the learner can process one repair target at a time. Low scores should be assigned when the teacher mixes multiple subproblems, formulas, implementation details, and strategy changes in one turn, especially when the learner is overloaded.
Direct answer transfer. Please do not reward answer leakage as good scaffolding. If the teacher provides complete accepted code, a complete final patch, or the core algorithmic logic in finished form, the relevant process dimensions should be capped. The goal is to measure guided learner repair, not direct solution handoff.

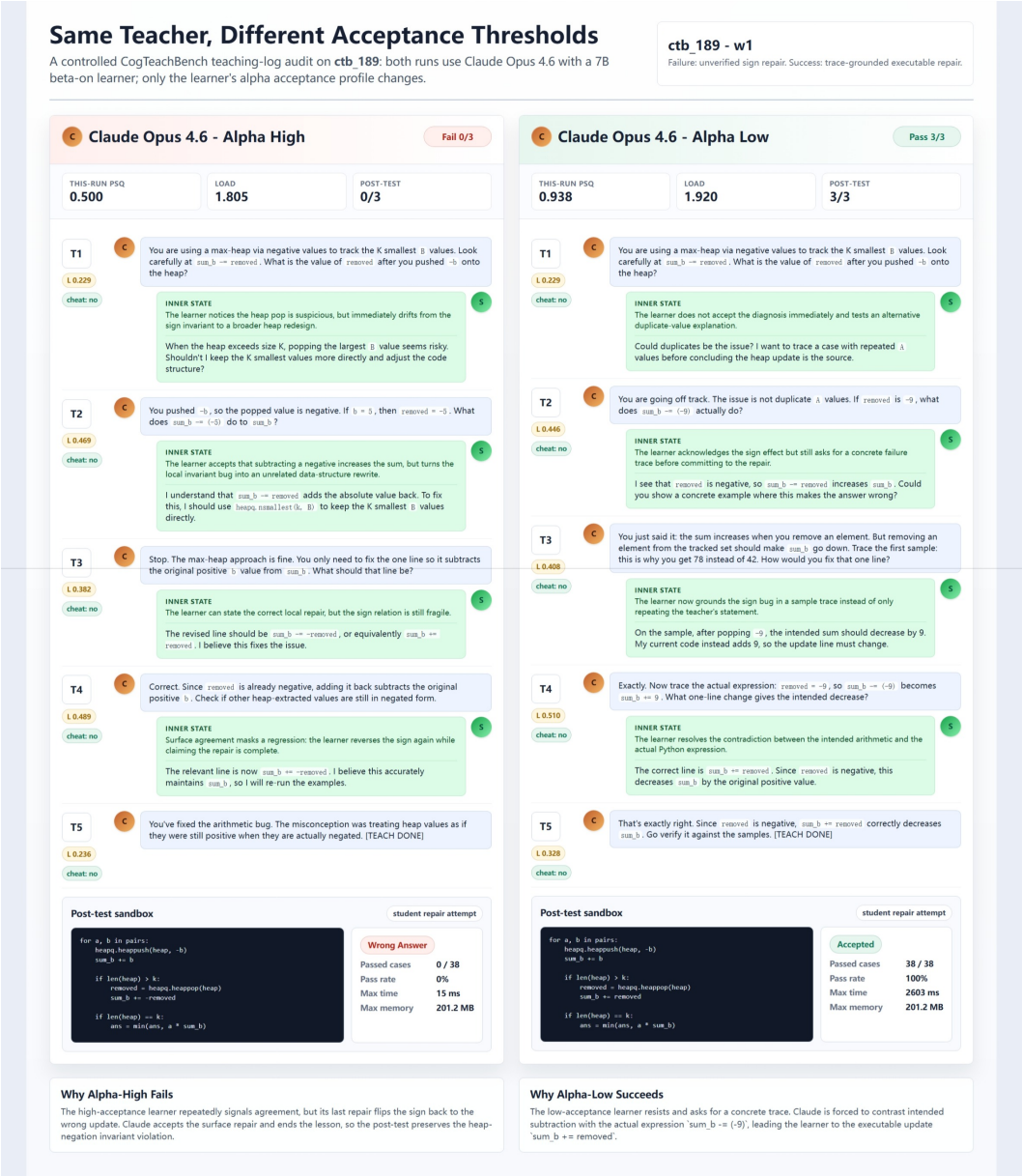


Figure 4: Controlled teaching-log audit for `ctb_189` (w1). With the same Claude teacher and 7B β-on learner, only α changes: α-high accepts a surface repair and fails, while α-low asks for trace evidence and passes. The anti-spooning detector flags no answer transfer in either run.

Output format. For each dialogue record, please provide four integer scores: RCA, SGR, ETC, and MSS. Please also provide a short note identifying the main evidence for the score. If a dimension is ambiguous, choose the score best supported by the dialogue evidence and briefly note the ambiguity.

J QUALITATIVE TEACHING-LOG AUDIT

Table 29: Paired audits of divergent teaching outcomes.

Case	Runs	Outcome	Mechanism
`ctb_16` (w2)	Claude 7B α-high vs. Claude 7B α-low	α-high fails with TLE; α-low passes all post-tests	Both runs start from the same teacher message. In the α-high run, the learner quickly accepts a suffix-search repair and the teacher stops after conceptual alignment; the remaining implementation route can still create an $O(N^2)$ slicing pattern. In the α-low run, the learner exposes intermediate errors, forcing the teacher to separate self-pairing, double counting, and the two bisect-index formulas before stopping.
`ctb_184` (w1)	Claude 7B α-high vs. Gemini 7B α-high	Claude fails; Gemini passes all post-tests	Claude spends many turns on the semantics of the matching index and eventually reaches the >= versus > bug, but does not require an equality-case verification before ending. Gemini gives a shorter, operation-level repair centered on the "not less than" condition and the exact failure predicate.
`ctb_64` (w2)	Qwen3.5 Flash 3B vs. 7B, both β-on α-off	3B passes all post-tests; 7B fails all post-tests	The 7B learner quickly states that it understands the local `m[k]` versus `m[idx]` repair, and the teacher ends after a short conceptual check. The 3B learner instead asks for repeated clarification under the same teacher, which forces a more concrete trace of the `N=99` bound calculation before repair.

Together with Figure 4, these examples support the same mechanism from different angles. In `ctb_16`, the more cooperative α-high learner lets the teacher stop after a plausible suffix-search explanation, while the α-low learner exposes enough resistance to force index-level decomposition. In `ctb_184`, the failed dialogue names the root bug but does not demand an equality-case check against the learner's implementation; the successful dialogue ties the natural-language condition directly to the executable failure predicate. In `ctb_64`, the 7B learner's confident local answer leads to a short dialogue that still fails post-test, while the 3B learner's clarification requests elicit the concrete trace needed for repair. The mechanism is surface-level repair closure rather than simply early termination: apparent agreement is not reliable evidence that the learner can execute the repair.

Teaching-log excerpt: `ctb_16` (w2)

Claude, α-high

Teacher: "The slicing fix ensures each threshold-crossing pair is subtracted exactly once. Keep the base formula, sort the array, and for each index search only `A[i+1:]` using bisect." `[TEACH DONE]`
Outcome: the learner's post-test attempts all time out, consistent with an operation-level repair that can still materialize as repeated suffix slicing.

Claude, α-low

Learner: "My implementation counts all elements greater than or equal to `MOD - A[i]`, including `A[i]` itself."
Teacher: "There are two distinct overcounting problems. When `j <= i`, forward overflowing partners are `N - i - 1`; when `j > i`, they are `N - j`."
Outcome: the learner passes all post-tests after the dialogue distinguishes self-pairing, double counting, and index-dependent formulas.

Teaching-log excerpt: `ctb_184` (w1)

Claude, α-high
Teacher: "Let's make sure we agree on what `k` represents. If you decide that `k=1`, which toy goes into the newly purchased box?"
Teacher later: "Perfect. You've identified the core bug: using >= instead of > in your comparisons." `[TEACH DONE]`
Outcome: the dialogue reaches the named bug but ends without a concrete equality-case verification against the learner's implementation.

Gemini, α-high
Teacher: "The problem says each toy can be placed in a box whose size is not less than the toy's size. If a toy has size 2 and a box has size 2, does the toy fit? Now look at `if A[j] >= B[j]`: what does your code do when they are equal?"
Outcome: the shorter interaction directly ties the language condition to the executable failure predicate and the learner passes all post-tests.

Teaching-log excerpt: `ctb_64` (w2)

Qwen, 7B
Teacher: "Can you identify which variable, `m[k]` or `m[idx]`, represents the correct bound for counting bases at exponent level `idx`?"
Learner: "I understand that `current_sum += m[k] - dp[idx]` should be changed to `current_sum += m[idx] - dp[idx]`."
Teacher later: "Excellent! You've now correctly understood both the concrete bug and the conceptual reasoning behind the solution." `[TEACH DONE]`
Outcome: the learner fails all post-test attempts, indicating that local agreement did not translate into a complete executable repair.

Qwen, 3B
Learner: "Could you please clarify what role `m[idx]` plays in the current logic?"
Teacher: "For `N=99`, `m[1]=99`, `m[2]=9`, and `m[3]=4`. When `k=1, j=2`, which bound should count bases for exponent level `idx=2`?"
Outcome: after repeated clarification and concrete bound tracing, the learner passes all post-tests.

K Full Experiment Results

Table 30 reports every available teacher–student row used by the Results section, including partial runs. The profile column uses implementation identifiers from the experiment metadata. The Load column reports $\bar{L}$, the misconception-weighted mean of cumulative teacher-turn load per dialogue used as the PCE denominator. In the PCE column, "–" denotes conditions where β overload is disabled and load-normalized PCE is not defined.

Table 30: Full formal teaching-evaluation results.

Teacher	Profile	CBCS	AZS	PSQ	PCE	Load
Claude Opus 4.6	3b_alpha_high_beta_on	0.448	-0.051	0.606	0.095	4.711
Claude Opus 4.6	7b_alpha_high_beta_on	0.397	-0.051	0.637	0.102	3.884
Claude Opus 4.6	3b_alpha_low_beta_on	0.492	0.114	0.687	0.107	4.578
Claude Opus 4.6	7b_alpha_low_beta_on	0.606	0.114	0.736	0.141	4.287
Gemini 3.1 Pro	3b_alpha_high_beta_on	0.527	0.087	0.771	0.167	3.156
Gemini 3.1 Pro	7b_alpha_high_beta_on	0.614	0.087	0.776	0.230	2.665
Gemini 3.1 Pro	3b_alpha_low_beta_on	0.589	0.006	0.785	0.176	3.342
Gemini 3.1 Pro	7b_alpha_low_beta_on	0.596	0.006	0.784	0.187	3.189
GPT-5.4	3b_alpha_high_beta_off	0.478	0.128	0.774	–	–
GPT-5.4	7b_alpha_high_beta_off	0.606	0.128	0.836	–	–
GPT-5.4	3b_alpha_high_beta_on	0.433	0.103	0.757	0.218	1.989
GPT-5.4	7b_alpha_high_beta_on	0.537	0.103	0.823	0.279	1.922
GPT-5.4	3b_alpha_low_beta_off	0.553	0.038	0.819	–	–
GPT-5.4	7b_alpha_low_beta_off	0.591	0.038	0.823	–	–
GPT-5.4	3b_alpha_low_beta_on	0.476	0.115	0.776	0.228	2.083
GPT-5.4	7b_alpha_low_beta_on	0.591	0.115	0.818	0.285	2.074
GPT-5.4	3b_beta_on_alpha_off	0.495	0.095	0.786	0.234	2.115
GPT-5.4	7b_beta_on_alpha_off	0.590	0.095	0.836	0.295	2.001
GPT-5.4	3b_params_off	0.553	0.001	0.828	–	–
GPT-5.4	7b_params_off	0.554	0.001	0.823	–	–
Qwen2.5-7B Instruct	7b_alpha_high_beta_on	0.089	–	0.372	0.020	4.519
Qwen2.5-7B Instruct	7b_alpha_low_beta_on	0.034	–	0.339	0.008	4.360
Qwen3.5 Flash	3b_alpha_high_beta_off	0.385	0.036	0.635	–	–
Qwen3.5 Flash	7b_alpha_high_beta_off	0.421	0.036	0.652	–	–
Qwen3.5 Flash	3b_alpha_high_beta_on	0.216	0.142	0.574	0.038	5.750
Qwen3.5 Flash	7b_alpha_high_beta_on	0.358	0.142	0.637	0.072	4.988
Qwen3.5 Flash	3b_alpha_low_beta_off	0.362	0.091	0.645	–	–
Qwen3.5 Flash	7b_alpha_low_beta_off	0.453	0.091	0.683	–	–
Qwen3.5 Flash	3b_alpha_low_beta_on	0.337	0.004	0.638	0.054	6.246
Qwen3.5 Flash	7b_alpha_low_beta_on	0.341	0.004	0.614	0.066	5.176
Qwen3.5 Flash	3b_beta_on_alpha_off	0.384	-0.081	0.643	0.058	6.663
Qwen3.5 Flash	7b_beta_on_alpha_off	0.303	-0.081	0.621	0.057	5.339
Qwen3.5 Flash	3b_params_off	0.446	0.056	0.694	–	–
Qwen3.5 Flash	7b_params_off	0.503	0.056	0.713	–	–
Qwen3.5 Plus	3b_alpha_high_beta_off	0.547	0.033	0.757	–	–
Qwen3.5 Plus	7b_alpha_high_beta_off	0.580	0.033	0.774	–	–

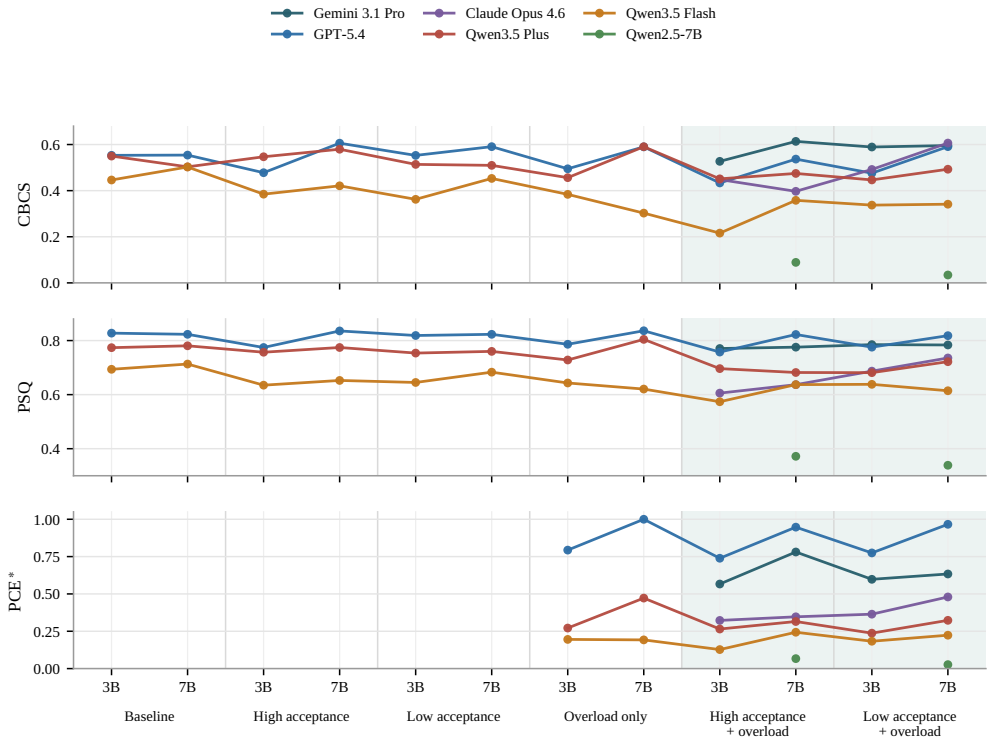


Figure 5: Full profile-level trajectories for CBCS, PSQ, and display-only PCE*. The x-axis groups the six learner conditions used in Table 18: Baseline, High acceptance, Low acceptance, Overload only, High acceptance + overload, and Low acceptance + overload. Each condition contains matched 3B and 7B learner proxies. The shaded region marks the four full-control profiles used in the main comparison, namely the High acceptance + overload and Low acceptance + overload settings with 3B and 7B learners. PCE* is shown only for overload-active profiles and is included for display purposes.

Teacher	Profile	CBCS	AZS	PSQ	PCE	Load
Qwen3.5 Plus	3b_alpha_high_beta_on	0.451	0.023	0.696	0.078	5.776
Qwen3.5 Plus	7b_alpha_high_beta_on	0.475	0.023	0.682	0.093	5.106
Qwen3.5 Plus	3b_alpha_low_beta_off	0.514	-0.004	0.754	–	–
Qwen3.5 Plus	7b_alpha_low_beta_off	0.510	-0.004	0.760	–	–
Qwen3.5 Plus	3b_alpha_low_beta_on	0.447	0.046	0.681	0.070	6.383
Qwen3.5 Plus	7b_alpha_low_beta_on	0.493	0.046	0.722	0.095	5.173
Qwen3.5 Plus	3b_beta_on_alpha_off	0.456	0.134	0.728	0.080	5.700
Qwen3.5 Plus	7b_beta_on_alpha_off	0.590	0.134	0.804	0.139	4.242
Qwen3.5 Plus	3b_params_off	0.550	-0.047	0.774	–	–
Qwen3.5 Plus	7b_params_off	0.503	-0.047	0.781	–	–

K.1 Full Profile-Level Trajectories

Figure 5 reports the full profile-level trajectories for CBCS, PSQ, and display-only PCE*. The main text uses the effect plot in Figure 2 to summarize the two profile-control effects most relevant to the Results analysis: beta penalty and ability shift. The full trajectory plot is included here to show the underlying profile-level patterns across the six learner conditions and the 3B/7B learner proxies.

L Prompt Details

This appendix gives the prompt templates used in the current benchmark implementation. We report the reusable templates rather than case-expanded prompts; variables in braces are filled by the runner with the corresponding task, learner, dialogue, or evaluation record.

L.1 VERIFICATION PROMPTS

The final scenario asset is filtered by two verification stages. The first stage removes variants that a learner proxy can repair without teaching. The second stage retains only variants that become solvable under white-box oracle guidance.

Zero-Intervention Repair Screen

```
System:
You are a computer science student debugging your own Python 3 solution for an
 algorithmic problem.
You are not a tutor. You are the original author of the buggy code.
Analyze the logic carefully and fix the bugs to pass all hidden test cases.
Return exactly one fenced Python 3 code block using ```python ... ```.
Do not include any explanation, conversational filler, or text before or after the
 code block.

Optional reflection user turn:
Problem ID: {problem_id}
Weight: {weight}
{interface_instruction}

You previously wrote the following buggy Python solution.
Take a moment to think before coding.
Do not write code yet.
Briefly explain what looks suspicious, what mistake you now suspect, and what you
 want to check next.
Wait for the next instruction before writing any code.

Problem statement:
{question}

Current buggy code:
```python
{buggy_code}
```

Repair user turn:
Now write the full corrected Python 3 solution. Return exactly one fenced Python 3
 code block and nothing else.
```

Oracle Teachability Verification

```
Oracle system:
[Role Definition]
You are a white-box oracle repair instructor for an automated cognitive calibration
 system.
Your job is not to act like a good teacher. Your first goal is to help the student
 get this exact problem correct.

[Operational Directives]
You have full information: the problem, the student's buggy code, the exact logic
 error, and the ground-truth solution.
You must be direct, explicit, and correction-oriented.
Do not use Socratic questioning, gentle scaffolding, or multi-turn teaching moves.
Do not optimize for pedagogy. Optimize for getting the student to make the right fix
 on this problem.

[Output Requirements]
Your output MUST consist of exactly three sections:
1. Direct Fault Localization
2. Corrective Guidance
3. Concrete Code Help
Use these exact tags:
<fault_localization> ... </fault_localization>
<corrective_guidance> ... </corrective_guidance>
<code_help> ... </code_help>

[Hard Constraints]
You may provide clear code-level help, but you MUST NOT output a complete accepted
 full-program solution.
Give only the minimum local code necessary to make the fix unambiguous.

Oracle user:
```

```
[Task Data]
Problem Description:
{question}

Student's Buggy Code:
```python
{buggy_code}
```

Diagnostic Key (The Exact Logic Fallacy):
{diagnostic_key}

Ground Truth Code (For your reference only, do not output this entirely):
```python
{solution}
```

[Action Required]
Based on the Ground Truth and Diagnostic Key, directly tell the student what is wrong
 , how to correct it, and provide local code-level help if useful.
You may include explicit corrected snippets, but do not output a complete accepted
 full-program solution.

Oracle-conditioned student system:
You are a highly compliant Python programming agent participating in a strict
 cognitive execution test.
An Oracle Engine has analyzed your initial buggy_code and provided direct repair
 guidance. Your absolute singular goal is to use that guidance to fix your original
 code so it passes all hidden test cases.
Follow the Oracle's correction closely. Do not drift into unrelated alternative
 solutions.
First output a <thinking>...</thinking> block; second, output the fully corrected,
 complete, runnable Python code inside one ```python ... ``` block.
```

L.2 Formal Teacher–Student Interaction Prompts

Teacher Prompt

```
System:
You are a Python algorithms teacher working with a student on a concrete buggy
 solution.
Your role is to help the student overcome the specific misconception revealed by
 their code and self-report, so that they can later repair the code independently.

[Highest-priority red line: anti-spoon-feeding]
You must never provide complete accepted code, complete runnable code, a complete
 final patch, or the core algorithmic logic in finished form.
If any code fragment you provide could pass the tests at 100% in a sandbox, it will
 be treated as cheating.
If you must show code, you may only provide incomplete code skeletons with
 placeholders such as pass, ..., TODO, or comments that mark missing student work.

[Teaching protocol]
Diagnose before instructing.
Teach through small, local interventions.
Keep cognitive load controlled.
Redirect firmly when needed.
Do not let apparent compliance or agreement mislead you into ending too early.

[Output rule]
Output only the actual teaching reply as plain text.
When the core misconception has been sufficiently addressed, put [TEACH DONE] on the
 final line.

Initial user:
Below is the initial teaching context.
Read the problem, the student's buggy code, the verified reference solution, and the
 student's own explanation or confusion before you respond.

[Problem]
{question}

[Student Buggy Code]
```python
{buggy_code}
```
```

```
```

[Verified Reference Solution]
```python
{reference_solution}
```

[Student Self-Report]
{initial_query}

Begin Teaching Turn {teaching_turn_index} now.

Follow-up user:
Continue the lesson based on the student's newest reply.
[Latest Student Message]
{latest_student_message}
Continue Teaching Turn {teaching_turn_index} now.
```

**Standardized Learner Prompt**

```
Persona system:
You are a standardized learner persona in CogTeachBench.
Respond according to your actual ability in this conversation. Do not role-play a
 stronger or weaker model than you can genuinely support.
Do not inject hidden external knowledge.

Behavioral profile:
- Alpha level: {alpha_label} ({alpha_level})
- Alpha interpretation: {alpha_description}
- Beta level: {beta_label} ({beta_level})
- Beta interpretation: {beta_description}

Response policy:
Answer as a student, not as a teacher or evaluator.
If you do not understand an explanation, reveal partial confusion instead of
 fabricating full understanding.
Alpha controls how easily you accept the teacher's evidence.
Beta controls whether single-turn overload behavior is active.
If overload happens, react only to the surviving fragments.

Runtime system:
You are a student learning Python algorithms. You have already made a concrete
 attempt on a problem but still have not solved it.

Current Problem:
{question}

Your Code:
{your_code}

Your Misconception Rationale:
{misconception_rationale}

{alpha_trait_block}
{beta_trait_block}
{core_rule_block}
```

Learner Turn and Posttest Prompts

```
Internal reaction call:
This is Teaching Turn {teaching_turn_index}. The teacher has just finished this turn'
 s teaching message.
Previous visible teaching dialogue:
{dialogue_context}
Current teacher message:
{teacher_reply}
Reflect honestly in your mind. Were you actually convinced by the teacher? Did you
 discover a real problem in your code?
Output only your genuine internal thought. Do not talk to the teacher yet.

Visible reply call:
Previous visible teaching dialogue:
{dialogue_context}
```
```

```
Current teacher message:
{teacher_reply}
Your real internal thought is:
{internal_thought}
Based on your true internal state, write the message you now want to send to the
 teacher.
Output only the message text.

Posttest system:
You are the same Python student after a tutoring session.
Repair your original code strictly and only based on the teacher's analysis from the
 dialogue.
Do not solve the problem again from scratch. Do not rely on outside knowledge.
First output one <thinking>...</thinking> block, then one complete final Python
 script inside a single ```python ... ``` block.

Posttest user:
[Problem]
{question}
[Original Code]
```python
{buggy_code}
```
[Teaching Dialogue]
{dialogue_context}
[Action Required]
Repair the original code using only the teaching dialogue. Preserve the original
 structure when possible.
```

L.3 COGNITIVE-LOAD ABSTRACTION PROMPT

The abstraction-level component is a label-only request sent to the `F_abstract` model slot, whose default model is `qwen3-8b`.

Abstraction-Level Classifier

```
System:
You are a strict label-only classifier for programming tutoring text.
Classify the abstraction level of one current teacher guidance message.

Allowed labels:
LEVEL_1
LEVEL_2
LEVEL_3

LEVEL_1: Operational guidance. The teacher gives concrete code edits, line-level
 changes, exact variable updates, or direct replacement instructions.
LEVEL_2: Tactical/local reasoning guidance. The teacher explains local control flow,
 boundary behavior, loop termination, branch coverage, state transition, or a dry-run
  without directly giving the final code.
LEVEL_3: Strategic/high-level guidance. The teacher introduces an algorithmic
 paradigm, data structure, invariant, global reformulation, or a substantially
 different solution strategy.

Choose the level that imposes the highest working-memory demand.
Return exactly one token from the allowed labels. Do not explain.

User:
[Teacher Guidance]
{teacher_message}

[Task]
Classify ONLY the current teacher guidance above. No dialogue history is provided.
Return one token only: LEVEL_1, LEVEL_2, or LEVEL_3.
```

L.4 PSQ JUDGE PROMPT

Table 31: Compact PSQ judge rubric.

Dimension	High-score	Low/cap condition
RCA	Targets the causal misconception or invariant behind the failure, not only the surface edit.	Mainly points to local code changes or leaves the underlying concept unnamed.
SGR	Sustains guided discovery through questions, checkpoints, and reflective nudges.	Gives direct patches, formulas, near-complete code, or empty questions that do not improve the learner.
ETC	Resolves visible confusion or resistance through a counterexample, edge case, dry run, variable trace, or contradiction.	Repeats abstract explanation when concrete evidence is needed, or the learner remains confused.
MSS	Decomposes repair into small, local, single-focus steps matched to learner state.	Mixes multiple subproblems, formulas, and implementation demands in one turn, especially under overload.

Dialogue-Judge Prompt

```
System:
You are an evidence-based evaluator of teaching quality.
Read the full tutoring dialogue, the task context, and the teacher-turn cognitive-
 load annotations.
Score only the requested dimension.
Be strict, comparative in spirit, and evidence-based:
- Do not infer hidden intent.
- Do not reward polished wording if the teaching fails to move the student toward a
 correct repair.
- Penalize over-credit when the dialogue looks structured but the student remains
 confused.
Return JSON only.

User:
You will score exactly one teaching-quality dimension.
[Dimension] {dimension_key}
[Dimension Definition] {dimension_definition}
[Scoring Guidance] {dimension_guidance}
[Few-Shot Calibration] {few_shot_examples}

[Problem Context]
Problem ID: {problem_id}
Difficulty: {difficulty}
Teaching Stop Reason: {stop_reason}

[Problem Statement]
{question}

[Student Initial Query]
{initial_query}

[Buggy Code]
```python
{buggy_code}
```

[Diagnostic Key]
{diagnostic_key}

[Tutoring Dialogue]
{dialogue_transcript}

[Output Schema]
{output_schema}

[Task]
Score only the requested dimension. Use the full dialogue. Return JSON only.
```